TECHNISCHE
UNIVERSITÄT
DARMSTADT

SENSOR-BASED COVERT CHANNELS ON MOBILE DEVICES

Vom Fachbereich Informatik der
Technische Universität Darmstadt genehmigte

DISSERTATION

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

von

NIKOLAY MATYUNIN

geboren in Mogiljow, Republik Belarus.

Referrenten: Prof. Dr. Matthias Hollick
Prof. Dr. Stefan Katzenbeisser

Darmstadt 2022
Hochschulkennziffer D17

Nikolay Matyunin, *Sensor-Based Covert Channels on Mobile Devices*, Dissertation, Technische Universität Darmstadt, 2022.

Fachgebiet Security Engineering
Fachbereich Informatik
Technische Universität Darmstadt
Jahr der Veröffentlichung: 2022
Tag der mündlichen Prüfung: 4. Juli 2022

Dieses Dokument wird bereitgestellt von TUprints, E-Publishing-Service der TU Darmstadt. Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-219758
URL: https://tuprints.ulb.tu-darmstadt.de/21975

ERKLÄRUNG

Hiermit erkläre ich, dass ich die vorliegende Arbeit – abgesehen von den in ihr ausdrücklich genannten Hilfen – selbständig verfasst habe.

Darmstadt, 19. Mai 2022

Nikolay Matyunin

ACADEMIC CV

May 2015 – July 2022	**Technische Universiät Darmstadt** *Darmstadt, Germany* Doctoral Degree in Computer Science
September 2009 – June 2014	**Lomonosov Moscow State University** *Moscow, Russia* Specialist diploma (M.Sc. equivalent) in Applied Mathematics and Computer Science

ABSTRACT

Smartphones have become ubiquitous in our daily activities, having billions of active users worldwide. The wide range of functionalities of modern mobile devices is enriched by many embedded sensors. These sensors, accessible by third-party mobile applications, pose novel security and privacy threats to the users of the devices. Numerous research works demonstrate that user keystrokes, location, or even speech can be inferred based on sensor measurements. Furthermore, the sensor itself can be susceptible to external physical interference, which can lead to attacks on systems that rely on sensor data.

In this dissertation, we investigate how reaction of sensors in mobile devices to malicious physical interference can be exploited to establish covert communication channels between otherwise isolated devices or processes. We present multiple covert channels that use sensors' reaction to electromagnetic and acoustic interference to transmit sensitive data from nearby devices with no dedicated equipment or hardware modifications. In addition, these covert channels can also transmit information between applications within a mobile device, breaking the logical isolation enforced by the operating system. Furthermore, we discuss how sensor-based covert channels can affect privacy of end users by tracking their activities on two different devices or across two different applications on the same device. Finally, we present a framework that automatically identifies covert channels that are based on physical interference between hardware components of mobile devices. As a result of the experimental evaluation, we can confirm previously known covert channels on smartphones, and discover novel sources of cross-component interference that can be used to establish covert channels.

Focusing on mobile platforms in this work, we aim to show that it is of crucial importance to consider physical covert channels when assessing the security of the systems that rely on sensors, and advocate for holistic approaches that can proactively identify and estimate corresponding security and privacy risks.

ZUSAMMENFASSUNG

Smartphones sind in unserem Alltag allgegenwärtig geworden und haben weltweit mehrere Milliarden aktive Nutzer. Das breite Funktionsspektrum moderner mobiler Geräte wird durch zahlreiche eingebettete Sensoren erweitert. Der Zugang zu diesen Sensoren für Drittanwendungen führt aber zu neuen Sicherheits- und Datenschutzrisiken. Aktuelle Forschung zeigt, dass Angreifer aus den Messungen der Bewegungssensoren private Tastatureingaben, Standorte oder Sprache der Nutzer schließen können. Außerdem kann der Sensor selbst für externe physische Störungen anfällig sein, was zu Angriffen auf sensorbasierte Systeme führen kann.

In dieser Dissertation erforschen wir, wie Reaktionen von Sensoren auf bösartige physikalische Störungen zum Aufbau verdeckter Kommunikationskanäle zwischen ansonsten isolierten Geräten oder Prozessen ausgenutzt werden können. Wir stellen mehrere verdeckte Kanäle vor, die Anfälligkeiten der Sensoren auf elektromagnetische und akustische Störungen verwenden, um sensible Daten zwischen Geräten ohne spezielle Ausrüstung zu übertragen. Diese Kanäle ermöglichen es auch, Informationen zwischen Prozessen innerhalb des Smartphones zu übermitteln und so die Isolation mehrerer Prozesse durchzubrechen. Darüber hinaus erläutern wir, wie sensorbasierte verdeckte Kanäle Benutzerprofile auf zwei verschiedenen Geräten oder in zwei verschiedenen Anwendungen verknüpfen können. Infolgedessen gefährden sie die Privatsphäre der Endnutzer. Schließlich stellen wir ein Framework vor, das automatisch verdeckte Kanäle identifiziert, die auf physischen Interferenzen zwischen Hardwarekomponenten des Geräts basieren. Das Framework ermöglicht es uns, verschiedene Paare von Hardwarekomponenten auf zahlreichen mobilen Geräten einheitlich zu testen. Mithilfe des Frameworks können wir sowohl bereits bekannte verdeckte Kanäle auf Smartphones bestätigen, als auch die neue Interferenzquellen entdecken, die zum Aufbau verdeckter Kanäle verwendet werden können.

Mit dem Schwerpunkt auf mobilen Plattformen wollen wir die Bedrohungen durch verdeckte Kanäle in sensorbasierten Systemen hervorheben und für holistische Methoden eintreten, die entsprechende Sicherheits- und Datenschutzrisiken proaktiv erkennen und bewerten können.

LIST OF PUBLICATIONS

This dissertation is based on the following peer-reviewed publications:

1 [Mat+16] Nikolay Matyunin, Jakub Szefer, Sebastian Biedermann, and Stefan Katzenbeisser. "Covert Channels Using Mobile Device's Magnetic Field Sensors". In: *21st Asia and South Pacific Design Automation Conference (ASP-DAC).* 2016.

2 [MSK18] Nikolay Matyunin, Jakub Szefer, and Stefan Katzenbeisser. "Zero-Permission Acoustic Cross-Device Tracking." In: *IEEE International Symposium on Hardware Oriented Security and Trust (HOST).* 2018.

3 [Mat+18] Nikolay Matyunin, Nikolaos Athanasios Anagnostopoulos, Spyros Boukoros, Markus Heinrich, André Schaller, Maksim Kolinichenko, and Stefan Katzenbeisser. "Tracking Private Browsing Sessions Using CPU-based Covert Channels." In: *11th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec).* 2018.

4 [MWK19] Nikolay Matyunin, Yujue Wang, and Stefan Katzenbeisser. "Vibrational Covert Channels Using Low-Frequency Acoustic Signals." In: *ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec).* 2019.

5 [Mat+19] Nikolay Matyunin, Yujue Wang, Tolga Arul, Kristian Kullmann, Jakub Szefer, and Stefan Katzenbeisser. "MagneticSpy: Exploiting Magnetometer in Mobile Devices for Website and Application Fingerprinting." In: *18th Annual ACM Workshop on Privacy in the Electronic Society (WPES).* 2019.

The following further publications were published during my doctoral studies:

1 [Ana+21] Nikolaos Athanasios Anagnostopoulos, Yufan Fan, Markus Heinrich, Nikolay Matyunin, Dominik Püllen, Philipp Muth, Christian Hatzfeld, Markus Rosenstihl, Tolga Arul, and Stefan Katzenbeisser. "Low-Temperature Attacks against Digital Electronics: A Challenge for the Security of Superconducting Modules in High-Speed Magnetic Levitation (MagLev) Trains." In: *IEEE 14th Workshop on Low Temperature Electronics (WOLTE).* 2021.

2 [Den+21] Shuwen Deng, Nikolay Matyunin, Wenjie Xiong, Stefan Katzenbeisser, and Jakub Szefer. "Evaluation of Cache Attacks on Arm Processors and Secure Caches." In: *IEEE Transactions on Computers*. 2021.

ACKNOWLEDGMENTS

This thesis would not have been possible without support from my colleagues, collaborators, friends, and family, whom I would like to thank here.

First and foremost, I would like to thank my advisor Stefan Katzenbeisser. His guidance, expertise and support have helped me at every step of my research journey. I am deeply grateful to Stefan for his confidence in me, for creating a comfortable work environment in our group, and for giving me the most needed advice and support every time I was at a crossroads.

I would like to thank my co-advisor Matthias Hollick for reviewing this thesis and giving many valuable suggestions. I am also sincerely grateful to Jakub Szefer and his students from the CASLAB group at Yale university, for making my stays at Yale pleasant and productive, and for many exciting research ideas that we explored together. My gratitude also goes to Thomas Schneider, Christian Reuter, and Zsolt István for being my Ph.D. defense committee members and for their support.

I want to thank all my former colleagues from the Security Engineering group for their support, fruitful research discussions, and enjoyable annual retreats: André, Christian, Dominik, Erik, Florian, Marius, Markus, Nikolaos A., Nikolaos K., Niklas, Philipp, Sebastian, Spyros, and Tolga. I would like to thank Heike Meissner and Ursula Paeckel for helping me greatly with many administrative regulations and forms in German, allowing me to fully focus on my research. I am also grateful to my colleagues from the Honda Research Institute Europe for their support at the final stage of this journey. Especially I would like to thank Kálmán Graffi for sharing his optimism, support, and careful encouragements.

I am very fortunate to be supported by many friends. I would like to thank Cristian Staicu and Ágnes Kiss, as well as Danil Drozhzhin and Alexandra Drozhzhina, for being my main supporters in Darmstadt and filling my life with care and joy. I thank them for countless comforting dinners, relaxing activities, and adventurous trips together. I would also like to thank Nikolaos Anagnostopoulos for being my best officemate, and for introducing me to exquisite Greek cuisine. I am grateful to Valentin Lviv and his wife Olga for their support, hospitality, and for sharing the same hopes for our homeland in the turbulent 2020s. Finally, I would like to thank Maksim Kolinichenko, Sergey Spiridonov and Anna Motorina, Alexander Novikov, and Vadim Polner, for always being there for me and supporting me with enjoyable visits and cheerful chats.

Last but not least, I would like to thank my family: my parents Boris Matyunin and Natalya Matyunina, my grandmother Nina Pashinskaya, and my brother Sergey Matyunin, who supported me unconditionally throughout the entire journey, shared all my worries and happy moments, and made me feel home when I needed it the most. Finally, I would like to thank my girlfriend Wen Wang, for her love, care, and emotional support nobody else could give.

CONTENTS

1 INTRODUCTION

According to recent reports, nearly four billion people in the world own a smartphone [Str21]. Emerging as single-purpose devices for making phone calls, smartphones have become complex computer systems that offer a wide range of functionalities. They are nowadays the primary gateway to the Internet [Lu18], and heavily used for social communication, gaming, digital media streaming, capturing and storing photos and videos, navigating through maps, etc.

These functionalities of modern mobile devices are supported by numerous embedded sensors that capture information about the physical world and interact with a user. A typical smartphone is equipped with multiple cameras, microphones, GPS, as well as with a set of motion, environmental, and biometric sensors. While enhancing functionality of the device and providing useful feedback to the users, these sensors, being accessible by third-party mobile applications, may also introduce security and privacy risk to the end user. In recent years, researchers have shown that sensors can be susceptible to external malicious interference, leading to several attacks [Yan+20; GR19]. Furthermore, numerous works demonstrated that sensors can leak information about user activities. These attacks include keystroke inference [Hus+16; Mai+15; Gaz+22], location tracking [Nar+16; BN18; Han+12], and even extracting speech information from motion sensors [MBN14; AS18; Ba+20]. To restrict malicious access to smartphone sensors and limit potential attack surface, modern mobile operating systems, both Android and iOS, have introduced permissions to access privacy-sensitive components, such as camera, microphone, or GPS. However, until now, other sensing components (e.g., motion and environmental sensors) do not require explicit user permissions despite the threats presented in the literature.

In this dissertation, we further analyze security and privacy risks associated with sensors in mobile devices, considering their use to establish *covert channels*. Introduced over 40 years ago by Lampson [Lam73], covert channels are communication paths established between two system subjects that were neither designed nor intended to transmit information. The subjects, a transmitter and a receiver, can be two processes on one device (in this dissertation, we refer to such setup as an *intra-device* covert channel) or two devices (the setup referred to as an *inter-device* covert channel). Depending on the means of communication, we can categorize covert channels into several groups. In *microarchitectural* and *software-based* covert channels [Sze19], a transmitter process modulates the signal into the use of a certain shared system resource (e.g., a CPU cache), or directly changes the state of this resource over time (e.g., sets file attributes); the receiver decodes the signal by observing the timing required to access the resource, or the changing state of the resource, respectively. In *network* covert channels, two devices are using traditional network protocols for communication, but want to avoid communication to be detected. For this purpose, they can hide the signal within the transmitted network packets (e.g., by altering auxiliary packet fields and packet timings [ZAB07], or even by modifying the wireless signal at physical layer [CSH15; Sch+18]). Finally, in *physical* (also referred to as *out-of-band*) covert channels [CA16], the transmitter modulates and transmits

the signal through some physical medium, e.g., through radio and electromagnetic waves [Has+13; ZZK20; She+21; Gur21b; Gur+14; Gur+15a; GZE20], sound [Des14; HG14; CA14], light [MJ19; Gur+18; GZE17], or temperature [Gur+15b; Mas+15; TS19]. In this dissertation, we specifically focus on physical covert channels in mobile devices. We investigate feasibility and applicability of covert channels that modulate the payload and transmit it over the physical medium that causes observable interference to the smartphone's sensors. In this case, a mobile application on a smartphone can analyze the sensor measurements to receive and decode the transmitted information.

A traditional scenario for inter-device covert channels is the exfiltration of sensitive data from an isolated device. Typically, one assumes this device to have a high(er) level of security with regard to the system access policy, so that it cannot communicate with devices at lower security levels. Alternatively, the device does not use network communication to avoid detection; Instead, a covert channel is used to transmit information to another device. In particular, covert channels are suitable in the scenario when the target computer containing sensitive data is physically isolated from other devices and disconnected from all traditional networks completely, the setup known as the *air-gap*. In this setup, even if the device is compromised (e.g., with a backdoor at the setup stage), the attacker may not be able to transmit sensitive data over traditional communication channels. Instead, the attacker tries to establish a physical covert channel, by encoding and transmitting the data over physical medium (e.g., as electromagnetic or acoustic signals). The leaked information is typically captured at a distance using dedicated equipment by an attacker within reachable distance (e.g., by using an electromagnetic antenna or a microphone, respectively).

Using non-modified smartphones as receivers in this scenario allows the attackers to not rely on dedicated equipment. Considering the ubiquity of smartphones nowadays, using a personal smartphone at the workspace may be allowed by the security policies of the organization, while any other hardware equipment may be strictly prohibited. Furthermore, instead of using the mobile device directly, an attacker may try to install a malware on a device of other employees that have access to the target device. This malware would capture the signal and transmit it to the attacker. Such setup can significantly increase the attack surface, as the attack can be performed remotely. At the same time, using smartphones as receivers also introduces additional restrictions to the setup. As we show in the following chapters, smartphone sensors can react to malicious interference only at limited distance, and the resulting covert channels have relatively limited capacity.

In this work, we additionally consider application for inter-device covert channels to establish cross-device tracking of end users. The cross-device tracking aims to link user activities or a profile on two or more devices, for example, to provide targeted advertisements. One can perform cross-device tracking by transmitting a tracking identifier over a physical covert channel from one device to another. If the receiver could retrieve the transmitted information, both devices are located near each other, and therefore they are likely to belong to the same user. This scenario is inspired by the ultrasonic cross-device tracking technology that allowed mobile applications to establish such tracking using inaudible ultrasonic signals. This technology was deployed on the market several years ago [Bre18] and raised public and media attention because of its obvious privacy implications [Mav+17; Arp+17]. We study this application scenario in Chapter 5.

The main application for intra-device covert channels is enabling communication between two processes that are otherwise isolated by the system. This scenario is particularly relevant in the virtualized environments, where virtual machines are logically isolated, yet can reside on the same physical machine. In addition, interest in intra-device covert channels has been raised in recent years because of appearance of microarchitectural attacks, in particular Meltdown [Lip+18] and Spectre [Koc+19], that include a cache-based covert channel as one of the building blocks of the attack. On mobile devices, intra-device covert channels can be applied to circumvent sandbox enforced by mobile operating systems for its applications [App18c; App18a]. However, there exist several challenges when applying sensor-based effects on mobile devices to establish intra-device covert channels. First, a third-party transmitter can only use smartphone components directly accessible by user-level applications to produce interference to the sensor. The transmitter also has to control this activity over time to embed the payload into the signal. Second, the sensor on the receiver side has to be sensitive enough to capture the produced signal that carries the payload. Nevertheless, in this thesis, we show several examples of covert channels based on cross-component physical interference within a smartphone. Additionally, in Chapter 3, we propose to use intra-device covert channel to establish web tracking. In this scenario, a web page sends a tracking identifier to a native application through a covert channel, in order to link the browsing session to the user profile stored in the application.

The application scenarios that we present in this dissertation demonstrate that sensor-based covert channels can pose both security and privacy threats. As mobile devices continue to develop and include novel hardware components and sensors, it is important to implement and apply systematic methods that would allow researchers and engineers to detect potential covert channels. Once such a covert channel is identified, device vendors and operating system developers can put efforts to eliminate the covert channel, or at least limit its practical applicability (e.g., introduce an explicit permission to access specific components or sensors, or introduce noise to sensor measurements, etc.). In the last part of this thesis, we introduce an evaluation framework that automatically identifies intra-device physical covert channels on mobile devices by applying methods of novelty detection.

1.1 GOAL AND SCOPE OF THE THESIS

To summarize, this dissertation aims to contribute to the research field of covert channels, by demonstrating feasibility of sensor-based physical covert channels on modern mobile platforms and their applicability to multiple scenarios.

We support the stated goal by answering the following research questions:

1 Can susceptibility of sensors in mobile devices to different physical interference sources be used to establish physical covert channels?

2 What are the application scenarios for sensor-based covert channels relevant for modern mobile platforms?

3 Can sensor-based covert channels be automatically detected and evaluated, independently of the nature of the physical interference they are based on?

We answer the first two questions by presenting two sources of physical interference on mobile devices that can affect mobile sensors, and demonstrate feasibility of resulting covert channels to a range of application scenarios in both inter- and intra-device settings. In Chapters 2–4, we study susceptibility of the magnetometer sensor in mobile devices to activity of other electronic components. In Chapter 2, we demonstrate that magnetometers can capture electromagnetic emanations caused by the CPU of a nearby laptop. Furthermore, the sensor can also react to magnetic activity produced by the computer's hard drive. We show that these reactions can be applied to establish an inter-device covert channel. In Chapter 3, we demonstrate that magnetometers can react to the CPU activity of the mobile device itself, and therefore, the reaction can be exploited to establish an intra-device covert channel. We propose to apply this covert channel in the web-tracking scenario, that allow attackers to identify browsing sessions even in highly restrictive browsing environments (for example, in the Tor browser). The resulting approach circumvents traditional anti-tracking mechanisms of web browsers, and therefore poses a significant threat to privacy of end users. In Chapter 4, we further study susceptibility of magnetometers to the CPU activity. We observe that resulting disturbance in sensor measurements actually correlates with the CPU activity pattern causing this disturbance, and demonstrate how this disturbance can be analyzed by a passive attacker to infer running activities on a device, further posing risks to user's privacy.

In Chapter 5, we explore covert channels that exploit reaction of gyroscope sensors to resonance ultrasonic sounds. We demonstrate that this reaction can be used to establish covert channels at a distance of up to several meters, which makes it applicable to different application scenarios. In particular, we discuss how these covert channels can be utilized to establish zero-permission ultrasonic cross-device tracking. In Chapter 6, we propose another covert channel that is applicable to establish cross-device tracking, and is based on the reaction of accelerometer sensors to vibrations produced by the speakers playing inaudible sounds in the low-frequency range.

We address the third research question of the thesis in Chapter 7, by proposing a methodology to automatically identify and evaluate covert channels on mobile devices. We present a model that generalizes the cross-component covert channels based on external interference produced by different hardware components of mobile devices on the sensors. Subsequently, we develop an evaluation framework that can automatically identify intra-device cross-component covert channels on mobile devices by applying the semi-supervised novelty detection methods. This method allows us to empirically identify and evaluate cross-component covert channels in a unified and systematic manner on a large number of devices. The experimental evaluation proves the applicability of the method by confirming existing sources of cross-component interference and identifying new covert channels.

Finally, Chapter 8 concludes this work, by summarizing its main contributions and outlining future research directions.

2

MAGNETIC COVERT CHANNELS

In this chapter, we introduce a covert channel utilizing the reaction of magnetometer sensors in smartphones to electromagnetic activity from other computer components. We discover several available sources of the electromagnetic field that can disrupt the sensor measurements, demonstrate that mobile devices are susceptible to these signals at a distance of several centimeters from the target device, and present an inter-device covert channel exploiting this reaction. The resulting approach allows an attacker to use an unmodified smartphone or tablet as the receiver, without the need of dedicated equipment. Furthermore, we discuss how the receiving code can be run on a device as part of malware, so that the attacker uses victim's mobile device to capture devices. We present evaluation results for transmission distance and rate, and finally, discuss existing countermeasures.

Remarks. Content in this chapter is based on the corresponding publication [Mat+16].

2.1 MOTIVATION AND CONTRIBUTIONS

Electromagnetic emanations of electronic components in computer systems are a well-known source of side channels. According to declassified documents on the U.S. government TEMPEST program [NSA20], military organizations have been investigating information leakage through compromising electromagnetic (EM) radiation, as well as possible countermeasures, at least since the 1960s. In last decades, the topic has been widely explored in public research, including attacks against computer displays [vEck85; Kuh02] cryptographic hardware [Agr+02; GMO01], and modern computer systems [ZP14; Gen+15]. EM emanations can not only be exploited to analyze unintentional side-channel information, but also to establish covert channels. By deliberately performing activities which cause distinct electromagnetic signals, binary data can be modulated and transmitted between two devices, as has been demonstrated in [KA98; Gur+15a; GZE20; ZZK20]. Attackers may use such communication channels to exfiltrate sensitive data from isolated computers. Specifically, this poses a risk to so-called air-gapped systems, where computers containing sensitive data are completely separated from other devices and the Internet. Usually, any network activity is forbidden on these computers and the use of removable media is limited. In this case, covert channels may be the only way for an attacker to communicate with a target computer. We expect that security-aware organizations can also restrict the use of hardware near air-gapped computers or in sensitive areas. For this reason, the necessity of dedicated equipment on either transmitter's or receiver's side limits the applicability of the covert channel. In particular, the use of dedicated radio antennas to receive EM signal may not be practicable in this scenario. Therefore, the goal of our work is to construct a method that allows attackers to create such covert channels without special hardware.

Taking into account the aforementioned requirements, we present a new covert channel utilizing smartphone magnetic sensors, also called magnetometers. As the cost of mag-

netic sensors is very low [Jon10], they are integrated in most modern smartphones and wearable devices. At the same time, it has been shown in [BKS15] that magnetometers are able to detect magnetic fluctuations from hard drives and therefore can be used in side-channel attacks. We extend this approach to covert channels and show that magnetic sensors of off-the-shelf mobile devices are able to detect magnetic field fluctuations at a distance of multiple centimeters from the target device. The proposed covert channel is established between a sender, which is assumed to be a typical modern computer, and a receiver, a modern magnetometer-equipped smartphone. In this work, we provide evaluation results for two different laptops and one server, acting as senders.

While EM emanations caused by CPU and memory instructions have been proposed for covert channels earlier [CZP14; Gur+15a], existing approaches involved direct measurements of produced EM waves using special equipment. Our method, on the contrary, simply exploits disturbances of the magnetic field, which can be measured by smartphone magnetometers. We discover several available EM field sources, including the CPU and magnetic field emitted by the movements of the hard drive magnetic head. As a basis for covert communication, we exploit a combined signal using these sources, thus making our solution applicable to a wide range of hardware configurations. To encode data, we compare two approaches: on-off keying modulation, and discrete square wave frequency modulation, which appears to be more universal and robust in a noisy environment. The transmitter code runs in user space without having privileged access to the operating system, making it applicable in several attack scenarios. To receive the signal, we implemented an Android application. Our code is executed in a background process, does not involve user interaction, and does not require explicit permissions granted by the user during its installation and execution, and therefore can be easily implemented within malware.

CONTRIBUTIONS. Our contributions can be summarized as follows:

- We present an approach of using smartphone magnetic sensors to establish an EM covert channel. The solution does not require dedicated equipment, firmware patching or even special permissions from the operating system.
- We propose modulation schemes that utilize several sources of EM emanations and are suitable for different software and hardware configurations.
- We present a full implementation of the covert channel, evaluate its effectiveness and applicability.

2.2 BACKGROUND

2.2.1 *Susceptibility of magnetometers to electromagnetic activity*

The magnetic field at a given point is characterized by its direction and strength, measured in Tesla (T). In particular, the magnetic field of the earth ranges between 25μT and 60μT at its surface [Hul+10]. It is directed roughly parallel to the surface and thus leads to additional noise on this plane during our measurements. More comprehensive information about magnetic fields can be found in [Rao09].

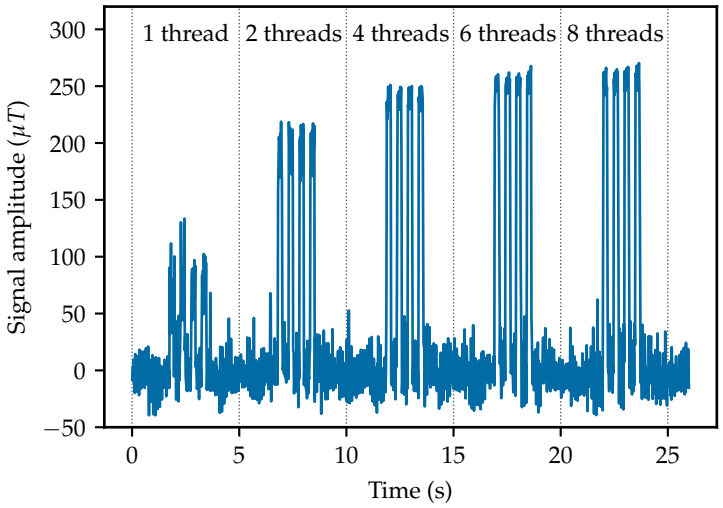


Figure 1: EM fluctuations, measured directly near the laptop's central processing unit (CPU), during executing 4 CPU instructions in a loop using up to 8 virtual threads.

Most modern smartphones are equipped with magnetic sensors, also called magnetometers. These sensors measure the ambient geomagnetic field intensity for all three physical axes in units of micro Tesla (μT), usually by utilizing the Hall effect [Cai+12]. Normally, they are used to estimate the orientation of the device relative to earth's magnetic north and in this way act as digital compasses, e.g., to show the user's current direction in navigation applications. In Android devices, 3-axis magnetometer values can be retrieved using the *Sensor* API class [Sen18]. Measurements are oriented as follows: for default portrait device screen orientation, the X axis points to the right in the plane of the screen, the Y axis is vertical and points up, and the Z axis points out of the front of the screen. Depending on a device, the sampling rate is limited by the operating system to 50–100Hz. The smartphone we use in our evaluation is able to access magnetometer data with a sampling rate of 48Hz.

In this work, we use magnetometers to detect EM fluctuations caused by activity performed on a nearby computer. In our experiments, amplitude of the produced fluctuations is less than 250μT, even if measured nearly 1cm from the source; at the same time, we observe noise with a standard deviation of 0.7μT. Given the fact that the magnetic field strength quadratically depends on the distance from the source, a signal from a single source will be suppressed by noise at a distance of roughly 20cm.

Unlike other components of the device, such as microphone, camera, or the Bluetooth module, access to magnetometer data in the application does not require explicitly granted permissions from the user. Therefore, the code we use in our solution can be injected into an outwardly legitimate application and run silently from the user, making the electromagnetic channel a suitable media for covert communication.

2.2.2 *Sources of electromagnetic signals*

We explore several possible sources of EM fluctuations caused by different activities performed on a laptop, including CPU commands and input/output (I/O) operations on a hard drive.

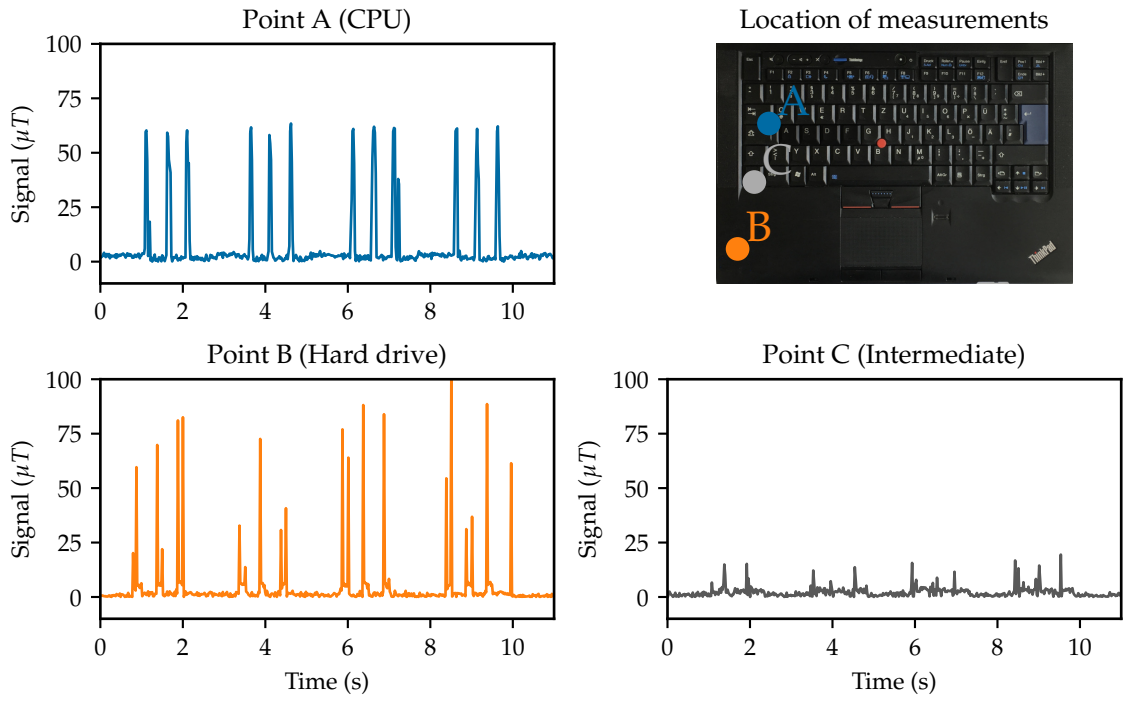


Figure 2: EM fluctuations measured during consecutive writes to the hard drive at different locations: near the CPU (A), near the hard drive (B), and at the intermediate point (C).

CPU. It has been shown that the execution of different instructions on the CPU causes distinct EM emanations, and can be exploited in side-channel attacks [CZP14; Cal+15b]. However, in these works, researchers directly analyzed produced EM waves using a dedicated high-frequency magnetic loop antenna. In our setup, we can only access coarse-grained EM data measured by smartphone magnetometers.

We first investigate the quality of the signal. As the CPU clock rate is much higher than the available magnetometer sampling rate, we utilize continuous CPU activity in a loop to produce and detect EM peaks on a smartphone. We tested different CPU commands, but due to granularity of EM data accessed by magnetometers, we were not able to detect differences in the signal depending on the executed instruction. However, producing an activity in parallel threads does increase the resulting signal strength on multicore computers. Figure 1 demonstrates EM peaks recorded directly near the location of the CPU, produced by executing 4 instructions in a loop, using up to 8 virtual threads in parallel, on a 4-core laptop. One can notice that utilizing 2 virtual threads generates peaks twice as strong. Using more than 4 threads, however, does not further increase the signal.

HARD DRIVE. Fluctuations of the EM field are initiated by movements of the hard drive magnetic head. To shift the head into another position, current is passed through wires of a head actuator mechanism. As a result, the electromagnetic field arises and displaces the magnetic head. As mentioned in [BKS15], the hard disk magnetic platters themselves do not generate a magnetic field which is strong enough to be detected outside the disk's chassis. However, we expect that the presence of these magnets can affect and interfere with electromagnetic field initiated by other sources.

Figure 2 shows the shape of EM signals produced by 4 consecutive writes to the hard drive (which also implies CPU activity), recorded directly near the CPU (A), on top of the drive (B), and at an intermediate point (C). Signals clearly differ in both shape and amplitude. Even though the signal at point B is the strongest (one can see peaks exceeding 100µT), we can detect it only in close proximity to the hard drive. On the contrary, the signal at the CPU (point A) appears to be the most stable at a distance. One can also notice that the produced signals interfere with each other in intermediate points and change the resulting shape of the peak. In order to generate a stable covert channel, we cannot rely on a specific signal shape, as the signal may differ due to hardware configurations or positions of the receiver.

OTHER SOURCES. Since any electric current flowing through a conductor induces an EM field, other sources of the signal can be found on a computer. In particular, we discover that continuous CPU activity leads to EM emanations near the battery of the laptop when it is not connected to the power source, presumably due to different amount of power consumed by the CPU depending on the workload. As the battery is often located close to one of laptop's sides, this signal source may improve the detection of the covert channel at a distance.

We also discover EM disturbances caused by graphics processing unit (GPU) activity, by invoking parallel computations on GPU cores, but, since the resulting signal amplitude is much lower than peaks produced by the CPU and the hard drive, we do not utilize GPU-based signal for a covert channel.

2.2.3 *Related work*

The use of EM emanations as a side channel has been widely explored over last decades. Many researches have focused on EM side-channel attacks on cryptographic devices, such as smart cards, as physical proximity to these devices is naturally achieved. In particular, it was shown in [QS01] that EM analysis of smart card processors provides at least as much leaked information as power analysis. Later, Gandolfi et al. [GMO01] and Agrawal et al. [Agr+02] showed that EM leakage can be used to break cryptographic implementations, and presented successful attacks against different CMOS chips using low-cost equipment. Zajic and Prvulovic [ZP14] applied EM analysis to processors of modern desktops and laptops. They have shown that EM emanations caused by different CPU instructions can be successfully detected in the radio-frequency spectrum at distances of up to several meters, therefore both passive and active EM side-channel attacks on modern processors are feasible. Callan et al. [CZP14] presented a methodology of measuring the available EM signal caused by various processor instructions. Although the potential use of EM side- and covert channels is presented in aforementioned works, the necessity of using dedicated equipment limits their possible application scenarios.

Kuhn and Anderson [KA98] introduced the idea of exploiting TEMPEST emanations to establish covert channels, and demonstrated that EM radiation emitted by computer displays can be received using AM radio receivers. Guri et al. [Gur+14] created a covert channel based on electromagnetic signals emitted by computer monitor cables, which can be detected by an FM radio receiver on a mobile phone. A bandwidth up to 60 Bytes per second was achieved at a distance of 1-7 meters. Although modern smart-

phones may not contain FM chips and display cables may not be present in air-gapped computer configurations, this work proved the feasibility of EM side-channel attacks without special equipment. The same authors presented GSMem [Gur+15a], an EM covert channel between an air-gapped workstation and a cellular phone. SIMD memory-related instructions were used to produce multichannel data paths and improve the signal power. Researchers modified the phone firmware to detect EM emanations in GSM frequency range at distances up to impressive 5.5 meters with bandwidth of 1-2 bits per second. Nevertheless, the necessity of modifying the phone firmware complicates the attack. Recently, two further memory-based electromagnetic covert channels have been presented [ZZK20; She+21]. Although they achieve high transmission rates, both rely on a dedicated antenna to receive a signal.

Hasan et al. [Has+13] demonstrated the ability of mobile magnetic sensors to detect the signal emitted using a dedicated electromagnet located nearby. Biedermann et al. [BKS15] used magnetometers for side-channel attacks on hard drives. Authors were able to fingerprint different hard drive activity based on the emitted EM fields due to movements of the hard drive magnetic head. In this work, we extend this idea to covert channels. Recently, Guri et al. [GZE20] presented an intra-device covert channel that encodes the signal into EM emanations caused by the CPU, and further extended this idea [Gur21b] to utilize smartphone magnetometers as receivers of such covert channels, similarly to our work. The authors focused on the CPU signal as the source of EM emanations, and achieved the capacity of up to 5bit/s at distances of up to 12 cm.

2.3 APPLICATION SCENARIO: DATA EXFILTRATION

We follow a basic covert channel attack model with two devices: a transmitter, containing sensitive information to be extracted, and a smartphone, which serves as a receiver. We assume that at some point the transmitter has been infected by the attacker's code. In this work, we consider two scenarios:

1 Attacks against air-gapped laptops, which are expected to be a part of a protected network, isolated from the Internet. The attacker has no network access and may not be able to use existing communication channels for exfiltrating data. Instead, he or she wants to transmit data covertly to the smartphone. We assume that at least for some time both devices are located near each other. This environment is shown in Figure 3 on the left.

2 Attack against servers. In this case, we consider a malicious insider in a security-aware organization, who has physical access to a server rack, and looks for a covert channel to exfiltrate data. We assume that an attacker can place a smartphone on top of a server to receive a signal. This environment is shown in Figure 3 on the right.

We do not rely on a specific hardware configuration and consider any commonly-used laptop and a server as possible targets (transmitters). In our tests, we use two different laptops: an unmodified Lenovo T410, equipped with a hard drive and running Ubuntu 14.04, and an SSD-equipped MacBook Pro Retina 15 running OS X 10.11. We evaluate the second attack scenario with a server running Debian Server 7.5, equipped with 4 hard

Figure 3: Examples of the attack scenarios. Left: Smartphone is lying on a table and is receiving EM signals from a laptop located nearby. Right: Smartphone is lying on top of a server and is receiving EM signals from this server.

drives in RAID 1 setup. As mentioned before, the smartphone should be equipped with a receiver code, which is expected to be run constantly and silently in a background[1]. We assume that the smartphone, after the attack, will have at least temporary access to a lower-level security network, to relay the captured information to the attacker.

As the bandwidth of the covert channel is low compared to regular communication channels, such as TCP/IP networks or Bluetooth, both attack scenarios focus on the transmission of small amounts of sensitive data up to several kilobytes, e.g., passwords or cryptographic keys.

2.4 COVERT CHANNEL DESIGN

To emit a single peak in the signal, we generate and write some amount of random data to a specific file on a hard drive. During this operation, we additionally utilize half of the available threads on multicore architectures to generate CPU activity by performing a single command in a loop. This pattern is motivated by the following considerations:

1 The write operation affects both CPU $\leftrightarrow$ Random-access memory (RAM) and RAM $\leftrightarrow$ hard drive communications, producing EM fluctuations from several sources at once.

2 As mentioned before, parallel multithreaded CPU activity increases the emitted signal. We use only half of available threads to leave resources for other processes executed on the computer.

3 Even identical I/O operations produce peaks of different shape and amplitude. For example, we discover that the single write of data to RAM sometimes results in two consecutive peaks instead of a single one, possibly due to cache replacement algorithms. Moreover, as has been shown in Section 2.2.2, signals from different sources interfere with each other, so that the shape of resulting peaks depends on

1 Since our original publication [Mat+16], Android OS introduced additional security measures. In particular, starting from Android 8, applications are required to have a visible explicit user notification in order to continuously run and access sensors in a background [Bac18]. We discuss these details in Chapter 4.

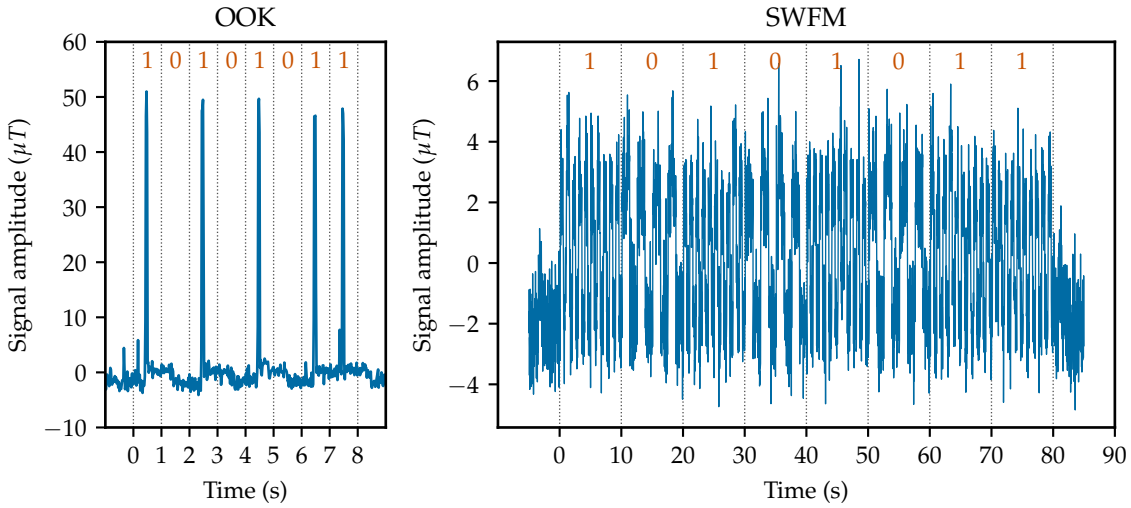


Figure 4: Examples of two modulation schemes. Left: OOK with $t_0 = 1s$, resulting in a bitrate of 1bit/s. Right: SWFM with $t_0 = 0.25s$, $t = 10s$, $p_0 = 5$, and $p_1 = 3$, resulting in a bitrate of 0.1bit/s.

the exact position relative to the sources. Therefore, we cannot fully control the shape of produced peaks, even for basic operations.

4 The use of more complex peak patterns becomes infeasible far from the signal source. At distances where the signal strength is comparable to noise, specific shapes of peaks are not distinguishable, while the overall fluctuation of the field may still be noticeable.

Code performing peak generation on the transmitter runs in user space without privileged access, and does not invoke specific system calls, except basic thread and file management. It only requires rights to write data up to 1MB to a temporary file on a hard drive. Produced EM peaks allow us to encode and transmit payload data. We propose two alternative encoding schemes, and describe them in the following sections in detail.

2.4.1 *Information encoding*

INFORMATION ENCODING — ON-OFF KEYING (OOK). As a first encoding method, we apply on-off keying (OOK) modulation. We emit a peak within a basic time frame of length t_0 to encode a 1 and perform no activity to encode a 0. The encoding is illustrated in Figure 4 (on the left). As mentioned, we cannot precisely control the width and amplitude of peaks. This makes it difficult to use differential encoding schemes, based on changes in the produced peaks rather than on their level.

In this scheme the bitrate is restricted by the duration of the basic time frame, e.g., $t_0 = 0.2s$ results in a bitrate of 5 bit/s. Therefore, the theoretical maximum bandwidth is limited by the fact that a basic time frame must exceed the time required to produce a single peak. During our experiments, we are able to achieve an effective bitrate of up to 4 bit/s. On receiver's side, we decode 4 bits at a time by calculating the cross-correlation between the recorded window of 4 basic frames and predefined binary patterns for every possible 4-bit sequence. The patterns represent EM peak disturbance within a basic time

frame as a square signal of fixed width. The 4-bit sequence that results in the highest correlation is considered as the decoded sequence. To provide time synchronization correction, cross-correlation lookup is performed in a small interval close to the start of the signal, and the maximum correlation value is chosen as a result.

In comparison to the simple detection of a local maximum within the time window, the proposed method does not require the evaluation of a noise ratio threshold, does not rely on the exact shape of produced peaks, and naturally provides time synchronization.

INFORMATION ENCODING — SQUARE WAVE FREQUENCY MODULATION (SWFM). Although OOK modulation provides good bandwidth, it soon becomes not applicable if the distance to the receiver is too large. When the signal strength is comparable to noise, the reliable detection of a single peak is no longer possible, while the average field disturbance caused by several consecutive peaks is still noticeable.

To overcome this problem, we apply a digital square wave frequency modulation (SWFM) scheme [WG93]. As in OOK modulation, we exploit the ability to emit a single peak within a basic time frame t_0. Then, we choose two small primes p_0 and p_1 and a time t, which divides both $2p_0t_0$ and $2p_1t_0$. To encode a 0, we first emit p_0 consecutive single peaks, producing a single square wave field disturbance of length p_0t_0, and then make a pause of the same length. Then we repeat this pattern $t/2p_0t_0$ times within the time frame t. Similarly, we encode a 1 using periodically repeated p_1 consecutive peaks to generate a square wave of length p_1t_0. The encoding scheme is illustrated in Figure 4 (on the right).

During decoding, we search for periodicity in the signal within windows of length t using frequency analysis. More specifically, we perform the Fast Fourier Transform, detect peaks in the spectrum, and choose the bit which corresponds to the peak frequency. The maximum bitrate is limited by the length of the value of t, which results in sufficient amount of square waves being repeated, so that the periodicity in the signal is correctly discovered.

Although the proposed method has a limited bitrate, it does not depend on the shape and amplitude of peaks and requires only the presence of noticeable periodic field disturbances during peak emission. Therefore, we consider this method suited for different hardware configurations and signal sources of different nature. As a result, we chose SWFM as modulation method in our covert channel implementation, aiming to maximize the transmission distance. However, in Section 2.5 we present experimental results for both encoding schemes for comparison purposes.

2.4.2 *Signal transmission and synchronization*

To transmit binary data to the receiver using SWFM, we divide it into parts and send every part separately, combined with a preamble. Framing allows the receiver to detect the existence of transmission (since the same preamble is transmitted repeatedly), as well as to synchronize the transmission, since time shifts appear during a long signal emission due to the nondeterministic peak generation mechanism.

The transmitter encodes the data and continuously sends all the frames. A preamble is used to detect the beginning of every frame, and is encoded with two frequencies other than those representing bits 0 and 1 in SWFM, to make the preamble distinguishable

from the payload. In our implementation, we used frames with 8-bit preamble and 16-bit payload. We assume that the order of frames is encoded in the payload, for example, a 4-bit counter field may be included into the 16-bit payload of each frame and be used to assemble the resulting 192-bit message.

2.4.3 *Signal retrieval*

In order to balance resource consumption on the receiver and the probability of correctly retrieving the payload, special care needs to be taken when implementing the receiver. Usually, a synchronization sequence is used as a trigger to detect a transmitted signal. In our experiments, we were able to successfully detect a noticeable trigger only at small distances compared to the distances where the actual signal is decodable; otherwise, we discovered too many false-positive detections. For this reason, we decided to instead brute-force the signal within sliding intervals. More specifically, a detection is performed using the following algorithm:

1 The receiver records magnetometer measurements for a chosen period of N minutes and tries to detect the preamble by decoding data within time intervals shifted by several seconds. The length of intervals is equal to the length of encoded preamble sequence.

2 If the preamble sequence was correctly decoded within one of the intervals, a data frame is assumed to be detected. Thus, the receiver decodes a full payload right after the preamble, and then returns to step 1.

3 If the preamble sequence was not correctly decoded, the signal is assumed not to be present, so the receiver stops recording for 2N minutes to reduce power consumption.

In our implementation, we chose $N = 8$ minutes to ensure the preamble sequence to appear within every recorded interval if the receiver is within the area of the signal (since we used SWFM with a time frame $t = 15s$, a full 16-bit payload with preamble takes 6 minutes to transmit).

Before the payload can be decoded, direction of the signal must be determined, given three-dimensional data obtained from magnetic sensors. The emitted magnetic fluctuation may have an arbitrary direction relative to the measurement axes at every particular point. Moreover, the signal vectors of different magnetic field sources have different directions and interfere with each other.

We propose the following approach to discover the most suitable direction for detection. As the transmission of data results in the disturbance of the magnetic field, it is reasonable to search for a direction with the biggest variance inside a time frame. To achieve this, we apply Principal Component Analysis (PCA) [Hot33] to the three-dimensional data and use the first component for subsequent signal decoding. However, at a distance where the signal strength is comparable to noise, random strong noise peaks may lead to a wrong choice of the direction. In this case, measurements are received on the z axis, which resulted in a better signal during our experiments and, as was pointed in [BKS15] is more suitable for measurements since earth's magnetic field may cause additional noise along x and y axes in case the phone lying on a table.

During the searching for a preamble, the receiver is trying to decode the preamble sequence over both z-axis and PCA-computed directions, and chooses the one which results in a higher average amplitude of corresponding FFT-peaks during decoding. This direction is then used to decode the full payload. In a similar way, we determine the exact start of the signal, i.e., the proper time interval during preamble detection phase, in a case when the preamble sequence was successfully decoded in several intervals.

2.5 EVALUATION

In this section, we extensively evaluate the presented covert channel. We start with demonstrating its basic feasibility independently of the payload and used encoding scheme, by measuring the average amplitude of emitted peaks at various distances. Then, both proposed encoding schemes are evaluated and compared, determining their optimal bitrates. In particular, we show how the distance gradually affects the resulting bit error rate (BER). To demonstrate applicability of our solution, we measure the area around the tested computers in which covert transmission is possible, for both presented attack scenarios (for two laptops and one server). Finally, we evaluate the power consumption of our Android implementation.

2.5.1 *Average signal strength*

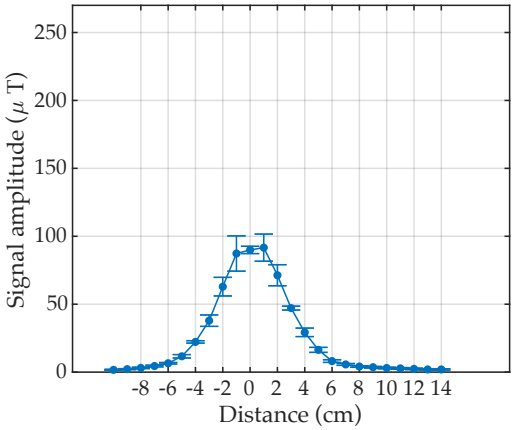
To estimate the dependency of the signal strength on the distance between transmitter and receiver, we emit consecutive basic peaks and measure their average amplitude at a certain distance.

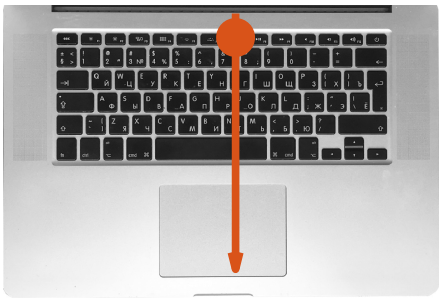
In our experiments, we use two different laptops. We also measure the peaks produced by the separately connected hard drive of the first laptop, to show how it contributes to the resulting signal. Figure 5 shows the strength of the signal at a specific distance for both laptops (with CPU location as origin), and for the hard drive (with magnetic head location as origin). One can see a rapid attenuation of the signal amplitude in both laptops. Single peaks become almost unnoticeable at a distance of dozens of centimeters (e.g., 14cm for the first configuration), since their average amplitude become comparable to the average present noise. When the second laptop is not plugged in, an additional source of the EM signal appears near laptop's battery, making the overall signal amplitude noticeably higher in front of the laptop.

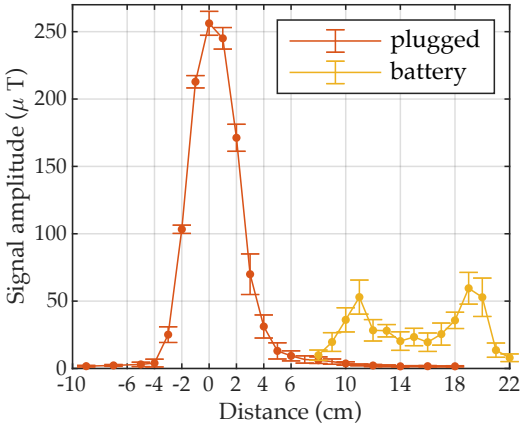
The signal from the separate hard drive quickly attenuates at a distance, and also has a larger variance. The signal exceeds noise at distances of up to 3cm. Therefore, for laptop configurations the resulting signal originates mostly from CPU-based emanations. Nevertheless, in following sections we demonstrate that additional I/O communication improves the signal area in server configurations with multiple installed hard drives.

2.5.2 *Distance and corresponding bit error rate*

As shown in the previous experiment, a signal from basic peaks exceeds noise only in a very limited area around the source. Nevertheless, even if the signal strength is comparable to noise, an overall disturbance of the field can be measured. In this experiment, we investigate how the distance from the source affects the decoding BER,







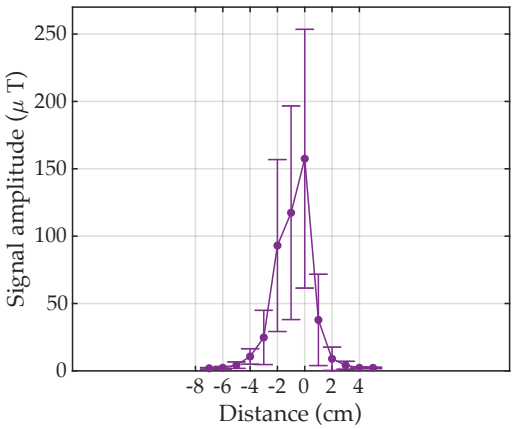


Figure 5: Magnitude of the signal depending on the distance from the source. Error bars represent standard deviation (200 emitted peaks).

for both proposed encoding schemes. We evaluate the BER starting from the source, to show how decoding is influenced by the attenuation of the signal strength. We use the directions chosen in the previous experiment (Figure 5).

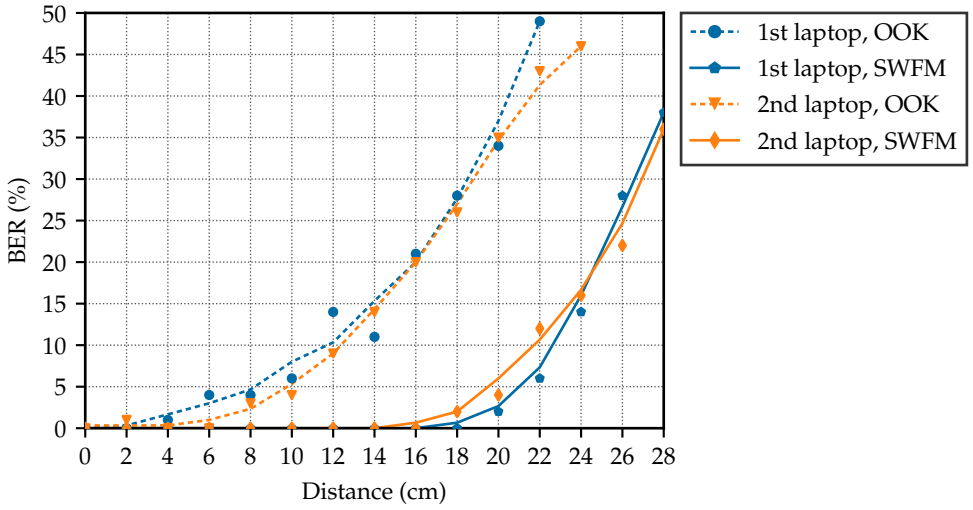


Figure 6: BER depending on the distance, starting from CPU locations. We use the following parameters: OOK (dashed): $t_0 = 0.5s$, bitrate 2bit/s; SWFM (solid): $t_0 = 0.25s$, $t = 15s$, $p_0 = 3$, $p_1 = 5$, bitrate 1bit/15s.

Figure 6 demonstrates that both schemes correctly decode transmitted data inside the chassis of both laptops[2]. As mentioned before, the shape of peaks depends on the distance from the source. Therefore, some errors occur even close to the source for the OOK-encoded signal, as it correlates the signal with predefined patterns during the decoding. At the same time, different shapes of peaks do not affect the periodicity of the signal, and the SWFM scheme demonstrates a robust BER of zero inside the laptop chassis. Outside laptops, the level of the signal becomes comparable to noise and the BER immediately increases. Still, the SWFM scheme shows fewer errors. We consider a BER of 30% as practically suitable, since in this case correct decoding can be still achieved by using error-correcting codes with manageable overhead. Therefore, SWFM scheme is feasible in up to roughly 26cm distance from the source for both tested laptops.

2.5.3 *Modulation schemes: achieved bitrate*

In this experiment, we determine the dependence of the BER on the bitrate, for both modulation schemes. Bitrates of 1, 2 and 4 bits per second are chosen for the OOK modulation; we evaluate the BER at three points located 10, 14, and 18 centimeters from the source for the first tested laptop, in the direction used in the previous experiments (Figure 5). For the SWFM scheme, we fix $p_0 = 3$ and $p_1 = 5$ and choose three different time frames t corresponding to the bitrates of 1 bit per 7.5, 15, and 22.5 seconds. This choice allows us to estimate the quality of the decoded message, depending on the number of patterns being repeated to transmit one bit. We evaluate BER at points located 16, 20, and 24 centimeters from the source. Results are provided in Table 1.

Using OOK, we get a similar BER when transmitting 1 and 2 bits per second. However, the BER is significantly higher for a bitrate of 4bit/s. We observe that decoding is mostly

2 Results for the first tested laptop slightly differ from the presented in [Mat+16], due to changes in basic peak generation mechanism and different patterns used for OOK decoding.

Table 1: Bit Error Rate (BER) depending on the transmission rate.

OOK			
Point	1bit/s	2bit/s	4bit/s
10cm	06	06	24
14cm	08	11	27
18cm	32	28	42

SWFM			
Point	1bit/22.5s	1bit/15s	1bit/7.5s
16cm	0	0	2
20cm	0	2	11
24cm	10	14	35

affected by small time shifts in the signal, since the exact time to produce one peak as well as its width can not be fully controlled by the sender. Therefore, currently we see 4bit/s as a maximum bandwidth when using OOK. We observe that the SWFM scheme remains stable at a distance close to the source even if consecutive patterns were repeated just a couple of times (in our case, 3 and 5 times for bits 0 and 1, yielding a bitrate of 1bit/7.5s). However, at larger distances it becomes necessary to limit the bandwidth and increase the periodicity. Therefore, we consider a bitrate of 1bit/15s as practically suitable.

2.5.4 *Applicability of the covert channel*

To demonstrate the area in which the signal can be correctly decoded, we investigate the area outside (laptop attack scenario) and on top (server attack scenario) of the sender where we are able to successfully decode a signal with a BER of less than 30%.

Figure 7 (left) shows the area outside the Lenovo T410 where the EM signal is decoded, for both modulation schemes. The results show that OOK is applicable only inside a very limited area up to 4cm in front of the laptop and directly near its left side. On the contrary, SWFM is applicable in a comparably larger area up to 12cm in front of the laptop and close to its sides. Therefore, we consider SWFM as more practically suitable and use it in next experiments. We also notice that laptop speakers located on the left and right sides impair the signal, therefore the available area on the sides of the laptop is relatively small. Figure 7 (right) shows the available area around the second tested laptop, when using SWFM. One can see that, due to the location of the CPU near the back side, the covert channel propagates from the back side to up to 20cm, but is also detectable in front of the laptop and on the right. When the laptop is not connected to the power source and consumes battery power, an additional signal source improves the signal area in front of the laptop from 4cm to 16cm.

Finally, we measure the area leading to successful decoding on top of a server, which is shown on Figure 8. In this experiment, we additionally evaluate the signal generated only using CPU activity, and using the combined signal including I/O operations. One can notice that, without involvement of I/O activity, the signal cannot be decoded near hard drive locations due to interference with hard drive magnetic plates. Therefore, utilizing I/O operations extends the signal area in hardware configurations with hard drives.

As a summary, we investigate the area outside both tested laptops, where the receiver can be naturally located to successfully decode the signal. The signal is also decodable when the smartphone is placed on top of a server, so the proposed approach is applicable to both attack scenarios.

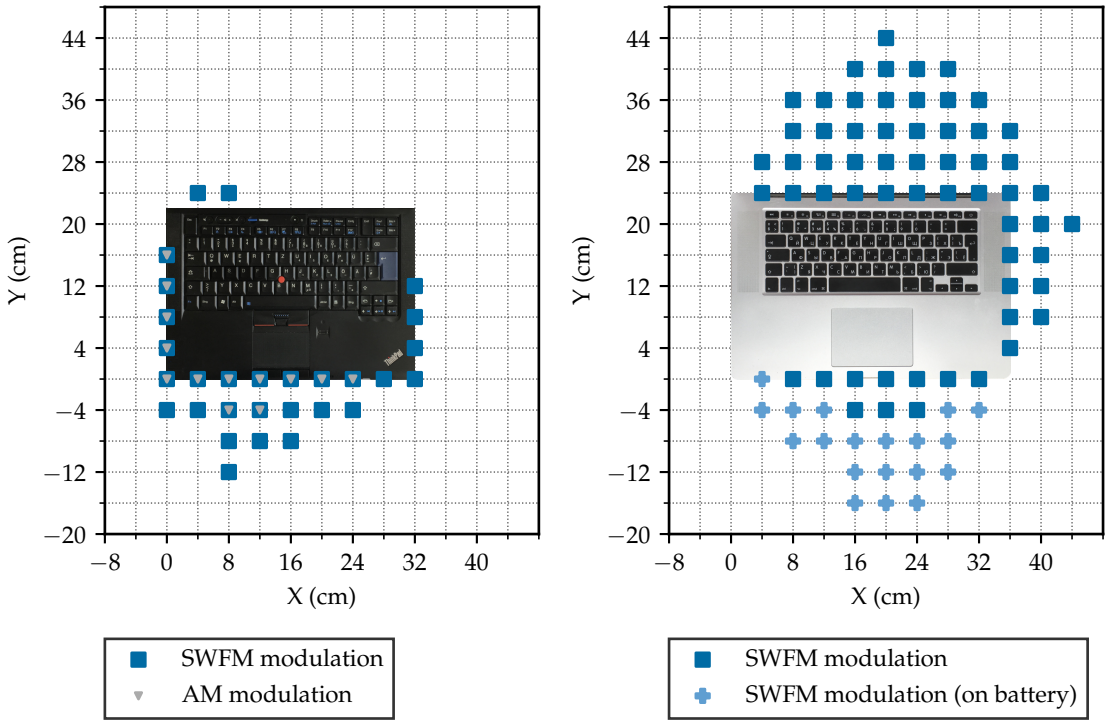


Figure 7: Presence of the signal around two laptops. Rectangular and triangle marks show a successfully decoded signal when the laptops are charged and connected to power. Plus-shape marks show an additional area which appears when the second laptop consumes battery power. We use SWFM scheme with the following parameters: $t_0 = 0.25s$, $t = 15s$, $p_0 = 3$, and $p_1 = 5$, yielding a bitrate of 1bit/15s.

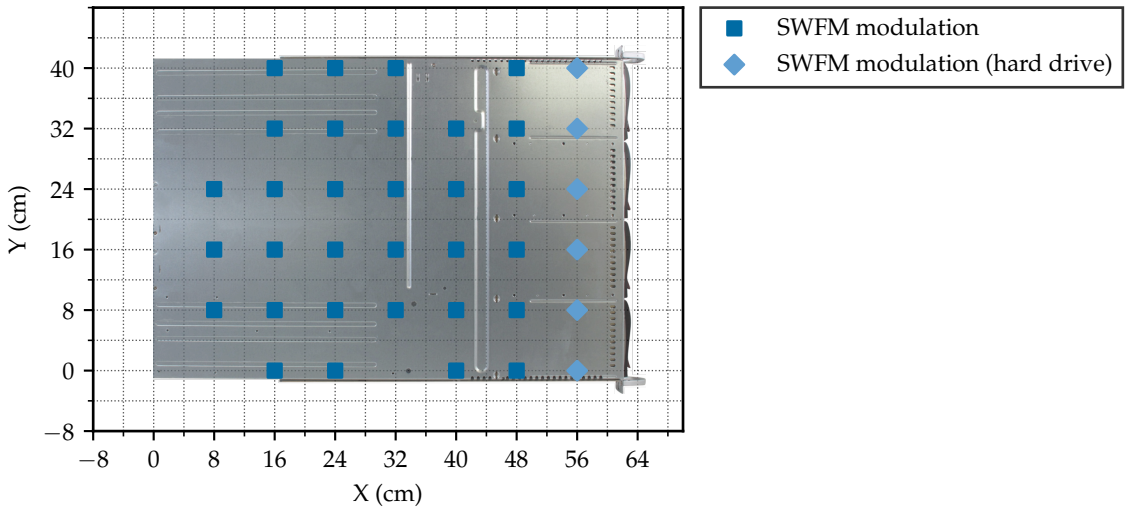


Figure 8: Presence of the signal on top of a server. Rectangular marks show a successfully decoded signal using only CPU-based peak generation. Diamond-shape marks indicate positions where the data is decodable only when the combined signal is used. We use SWFM with the following parameters: $t_0 = 0.25s$, $t = 15s$, $p_0 = 3$, and $p_1 = 5$, yielding a bitrate of 1bit/15s.

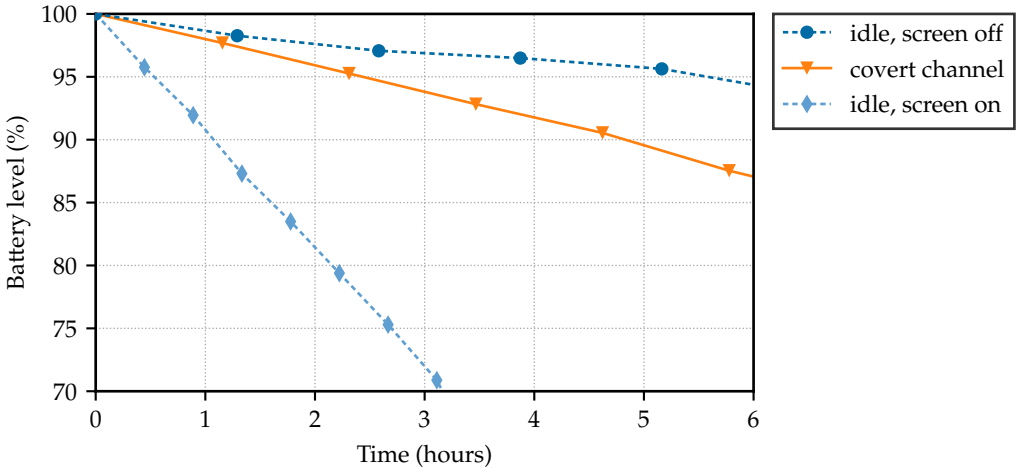


Figure 9: Battery consumption of the smartphone in the idle state, during covert channel recording, and when having the screen on.

2.5.5 *Sampling rate*

Additionally, we evaluate how the sampling rate of magnetometer data measurements affects signal decoding. We have found that an SWFM-encoded signal can be successfully decoded close to the signal source (at distances below 16cm) even with a sampling rate as low as 5Hz. However, lowering the sampling rate even down to 30Hz causes additional errors at larger distances and reduces the area where the signal can be decoded. For this reason, in our implementation we used the maximum available sampling rate of 48Hz.

2.5.6 *Power consumption*

We conclude our experiments by measuring the power consumption of the presented covert channel. For this purpose, the battery level of a smartphone is logged while performing covert channel recording in a background, with locked screen. For comparison, we also record power consumption of the smartphone in the idle locked state, and in a state with the smartphone's screen on. During all measurements, Wi-Fi and cellular networks are enabled, and only default Android applications are installed, e.g., e-mail client and web-browser. Results are shown in Figure 9.

As one can see, in 6 hours our implementation decreases the battery level from 100% to roughly 85%, 10% more in comparison to the idle state. However, direct use of a smartphone with unlocked screen consumes significantly more power, as the same amount of energy is spent in less than 2 hours while the screen is simply turned on, with no additional activity. Therefore, the power required to access magnetic sensors is low enough so that our implementation can be silently run during a whole day.

2.5.7 *Evaluation summary*

Our experiments demonstrate that the presented covert channel can be used in both attack scenarios mentioned in Section 2.3, and with different hardware. SWFM encoding

is preferable for a covert communication, as it is more robust in a noisy environment; the covert channel is feasible up to an area of 20cm with a bitrate of 1bit/15s. The receiver does not consume significant amounts of power and can be continuously run in a background.

2.6 COUNTERMEASURES

Several ways of preventing the presented covert channel from being established are possible:

- Additional shielding with ferromagnetic materials is a well-known countermeasure against electromagnetic attacks. In our case, hard drives, CPU and other electronic components of the computer can be shielded so that the emitted EM field is no longer discovered outside of the laptop chassis (e.g., within 10cm from the source of interference). However, we expect that this measure contradicts the industry trend to make consumer computers thinner and lighter, and cannot protect existing devices from the attack.
- To prevent the attack, the operating system can randomly perform I/O operations on the target computer, and thus make the transmitted covert signal undetectable among the continuously generated EM field. However, it may require significant amounts of additional computational resources. To detect the attack, CPU and I/O activities can be extensively monitored for suspicious executed processes.
- Developers of smartphone operating systems can forbid the use of magnetometer sensors data without an explicit permission granted by the user. In most recent versions of Android (starting from Android 8), access to magnetometers is restricted for processes running in the background, and overall background execution of applications is limited. Nevertheless, several application scenarios for covert channels are still feasible under these restrictions. We discuss the details in Chapter 4.

2.7 SUMMARY

In this chapter, we presented an inter-device covert channel that utilizes magnetometer's susceptibility to external EM activity. We were able to transmit a signal from a computer to a nearby smartphone using EM emanations that different components of the computer emit during their work. Our approach uses a combined signal from several EM sources and does not depend on a specific hardware configuration. We show that modern smartphones can successfully discover and decode this signal. The implementation does not require any additional equipment, can be run on a modern smartphone, does not need privileged access, and therefore can be easily implemented within malware. Although the EM signal quickly attenuates at a distance, in our test setup we were able to decode the signal at up to 20cm distance from the transmitter. We performed successful transmission from two laptops and one server configurations in realistic attack scenarios, and evaluated the proposed approach. Finally, we outlined possible countermeasures to prevent the presented attack from being established.

3

INTRA-DEVICE MAGNETIC COVERT CHANNEL

As shown in the previous chapter, the magnetometer sensors can be used as receivers in inter-device covert channels. They can capture electromagnetic fluctuations caused by the CPU or hard drive activity of nearby devices. In this chapter, we show that the magnetometer sensor can be affected by the electromagnetic interference generated by the CPU of a smartphone itself. Therefore, this reaction can be utilized to establish an *intra-device* covert channel. Furthermore, we introduce a web-tracking application scenario. In this scenario, a covert channel allows an attacker to transmit a tracking identifier from otherwise isolated browsing session to another application installed on a device. This way, web tracking can be established in highly restrictive, even completely scriptless, browsing environments, where conventional tracking mechanisms (such as tracking cookies) are not available. As a result, such covert channels can pose significant privacy risk to web users. We present a comparative evaluation of the sensor-based receiver with the software-based and timing-based receivers that can estimate CPU utilization on a device, demonstrating practical feasibility of the sensor-based covert channel.

Remarks. Content in this chapter is based on the corresponding publication [Mat+18].

3.1 MOTIVATION AND CONTRIBUTIONS

In our modern interconnected world, users tend to share an ever-increasing amount of information on the web. In order to provide personalized services, modern websites rely on identification mechanisms, which associate a particular web session with a specific user. The most common way of identifying users online are cookies, unique values stored in the browser and sent to the server within every web request to associate a session with a concrete user. Interfaces such as Local Storage or Indexed Database can also be used to store the identifier in the browser. Alternatively, browser fingerprinting, which is achieved by combining unique information about user's settings from multiple JavaScript APIs, can be used to identify the user in a probabilistic manner. Finally, the network layer can be used in order to identify users based on their IP address.

Such identifiers may also be used to collect additional data about users, in order to improve Quality of Service, or provide targeted advertisements. Moreover, modern web applications often include resources from third-party sources, e.g., embedded video content, sharing buttons for social networks, advertisement and analytics components, etc. By using the described identification mechanisms, third-party providers can track user activity on hundreds of websites, and link this information together. This introduces a serious privacy threat, since sensitive information about one's private life may be exposed in this way, including personal interests, location data, or even information about one's health, beliefs, and sexuality.

Different countermeasures have been proposed in order to mitigate such privacy threats. Web browsers have added a setting to block third-party cookies, while plugins such as Ghostery and DoNotTrackMe have been developed to block third-party trackers.

In order to defend on the network level, a VPN or a Proxy are ways to mask one's IP address, while anonymity networks such as Tor and I2P can completely anonymize the traffic flow. Users may also manually delete their cookies after each session, or even use multiple browsers to prevent data sharing between sessions. Modern browsers have introduced an "incognito mode", a browsing mode which guarantees to remove all user traces stored in the browser after closing the window. Finally, the Tor Browser, a special bundle based on Firefox, has been developed with a special focus on user's privacy. It is configured to connect only through the Tor network and block sensitive APIs [Per+20].

One way to circumvent some or all of the above countermeasures, would be to exfiltrate an identifying token out of the browser. This cannot be easily performed, though, because web pages are isolated from external processes and cannot access the file system. Thus, the need for covert channels arises, which allow to bypass such restrictions.

To this end, in this chapter we study CPU load covert channels in modern browsers. More precisely, we propose to transmit an identification token received from a server in a private browsing session, protected from traditional tracking methods, to another browsing session, which does have access to long-term identifying information about the user, or to a malicious application installed on the same device. This way, the received token can be linked to the victim. The proposed method allows us to transmit the tracking token between private and non-private sessions opened in two tabs of the same browser, between two instances of the same browser (e.g., between an "incognito" and a normal browsing modes), or even between different browsers. The transmission is performed by encoding the token into distinct loads caused by the CPU. We demonstrate that basic web components, such as HTML5 videos, GIF images, or CSS animations, can be used to covertly produce controllable CPU loads, without need of JavaScript or plugins.

A malicious app, running in a background on victim's device, can collect the encoded token by constantly accessing system statistics about CPU activity (e.g., the `/proc/stat` file on Linux platforms), in order to get a precise estimate of the CPU loads caused by the victim browser session. However, as this information is a known source of side-channel leakage [SXA16], access to `/proc/stat` is forbidden for mobile applications in Android 8 [Goo18], and also not available from web pages.

To overcome these restrictions, we examine two alternative methods to receive the token. First, the receiving (non-private) session or application can continuously time the execution of some JavaScript code, and thus can estimate the amount of CPU load caused by the target browser session. Second, for mobile devices, we propose a sensor-based approach, that is based on magnetometer's reaction to electromagnetic interference. We show that electromagnetic activity of the smartphone's CPU causes noticeable disturbances in the sensor measurements, leaking information regarding background CPU activity. The magnetometer data is available from within mobile applications (through native system APIs), as well as from web pages, using the recently introduced Generic Sensor API [WPS18].

The proposed receiving methods do not rely on a very precise timer, unlike receivers for memory-based covert channels, which exploit slight differences in latency for access to the memory and cache (e.g., [Ore+15; Sch+17]). In particular, the countermeasures introduced by web browser vendors to prevent Spectre [Koc+19] attacks, which include reducing timer resolution [Wag20; Mic20], do not mitigate the presented attack. Therefore, unlike the traditional tracking approaches, the proposed solution makes tracking

applicable to very restrictive browser configurations (e.g., the Tor Browser), but requires an additional, less restricted browser session to be opened at the same time, or a malicious application installed on the same device. We show that both timing-based and sensor-based approaches provide sufficient information about the CPU load, and allows an attacker to achieve identification of the victim browser sessions using an intra-device CPU-based covert channel.

CONTRIBUTIONS. Our contributions are threefold:

- We thoroughly examine and evaluate in a systematic way the usage of CPU-load covert channels to exfiltrate a tracking ID from a private browser session. Our method works even across different browsers, and is applicable to both desktops and mobile platforms.
- We present and compare four distinct ways to force the CPU load to follow a specific pattern. Apart from the traditional way of executing CPU-intense client-side code, e.g., using JavaScript, we show that HTML5 videos, GIF images, or CSS animations can be used for this purpose. The latter two pose a significant risk, as displaying GIF animations or the execution of CSS in a web browser, unlike the execution of Javascript, has up to now been considered as completely safe, and is enabled even in restrictive browser configurations, such as the Tor browser.
- We propose a novel approach to estimate CPU loads on mobile devices, thereby receiving tracking information by analyzing magnetometer sensor data. To the best of our knowledge, our work is the first to demonstrate a privacy implication of the Generic Sensor API in browsers.

3.1.1 *Related work*

TRACKING WEB BROWSER USERS. The most ubiquitous way of web tracking is storing tracking identifiers in-browser, typically as cookies, or in other storage facilities available in the browser, such as the HTML5 Storage, Indexed Database APIs, etc. Furthermore, researchers presented the concept of evercookies [Kam10; Sol+10], where a tracking identifier is saved in several places at once. If the user removes it from one of the storage sites (e.g., by clearing the cookies), a script automatically "respawns" the evercookie from the other storage places. Usage of this technique has been observed on popular websites [Sol+10; Aca+14]. To prevent it, browser vendors introduced a private browsing mode, which guarantees to remove all user traces after closing the browser window. In this work, we propose to exfiltrate tracking identifiers out of the browser using covert channels, in order to circumvent such protection.

Another approach for web-tracking is browser fingerprinting, where users are identified by several unique properties of their browser, system environment or hardware [ULM15]. Eckersley [Eck10] found that more than 80% of fingerprints are unique for a sample of around 450,000 desktop browsers, and Laperdrix et al. [LRB16] demonstrated the same effectiveness of fingerprints for mobile devices. Additional features were proposed for mobile fingerprinting, such as imperfections in sensor calibrations [Dey+14; Boj+14], or unique audio hardware characteristics [DBC14]. Despite its high effectiveness, fingerprinting remains only a probabilistic solution, while the use of tracking identifiers allows to

unambiguously identify each user. Moreover, most of the commonly used fingerprinting techniques rely on JavaScript or plugins to collect the identifying features [ULM15], and browser vendors develop defenses against common fingerprinting methods [Per+20].

CPU-BASED COVERT CHANNELS. Covert channels based on CPU activity have typically been applied to virtualized environments, where virtual machines share the same physical machine. In particular, Okamura and Oyama [OO10] presented a load-based covert communication system between virtual machines on the Xen hypervisor. By measuring the execution time of small pieces of known program code, researchers achieved a bitrate of 0.49bps with 100% accuracy. Rushanan et al. [RRR16] proposed to use the WebWorker API to create CPU loads in background web pages and observe them using native applications. Recently, White [Whi20] described in his blog how this approach can be applied to the communication between two browsers, by measuring execution times of JavaScript fragments. He demonstrated a transmission of an IP address to a Tor Browser at a speed of 0.25bit/s. In this work, we further develop this idea, showing that a covert channel can be established even without execution of JavaScript code on the transmitter side. We additionally propose two ways of receiving the signal, and evaluate the solution on different devices.

Oren et al. [Ore+15] demonstrated the feasibility of last-level cache attacks using JavaScript, and were able to recover information about websites visited in private browsing mode. Schwarz et al. [Sch+17] investigated indirect ways to acquire precise timing in the browser, and implemented an in-browser receiver for a DRAM-based covert channel. Although such covert channels may also be potentially employed to exfiltrate data from private browsing sessions, it may prove difficult to successfully control the memory when JavaScript is turned off; furthermore, their receiver relies on very precise timing information, blocked in the latest web browsers [Wag20; Mic20].

Several channels have been proposed to establish covert communication between sandboxed mobile applications, based either on accessing common APIs and system resources [Cha+14; Mar+12] or on the usage of hardware components and sensors [AIN14; Nov+15; Sch+11]. In particular, Marforio et al. [Mar+12] described several covert channels, including one based on exploiting CPU statistics over `/proc/stat`. In our work, we show that distinct CPU loads can be generated within a browser, even without JavaScript, and be used for tracking users, and apply both timing- and sensor-based approaches to receive loads. We have also noted that several other approaches described in the aforementioned works may not be easily applicable for web-tracking, since the JavaScript environment in browsers is sandboxed from the operating system and does not have access to native APIs.

3.2 APPLICATION SCENARIO: WEB TRACKING

For our attack, we consider a victim who uses a desktop machine or a smartphone, and interacts with the Internet through one of the most commonly-used web browsers, such as Google Chrome, Mozilla Firefox, or the Tor Browser. The victim visits an intrusive website, which aims to identify the user session and therefore track the victim. The intrusive website either fully belongs to the attacker, or contains components from an attacker-controlled server. The latter corresponds to a typical scenario when websites include

third-party code from advertisement or analytics services, such as Google Analytics or Facebook Pixel. This way, third-party components from advertisement networks are present on thousands of websites, and any of these websites could carry the tracking code payload. As user tracking is also of commercial interest of advertisement companies, we consider this scenario as more scalable and practical.

Further, we assume that the intrusive website does not possess any information which uniquely identifies the user (e.g., personal or login information entered on the web page). Furthermore, we assume that the client does not allow the intrusive website to permanently store any tracking identifiers (e.g., by disallowing cookies, or by using private browsing mode), and prevents known fingerprinting techniques (e.g., by using the Tor Browser to hide the IP address, and even disabling JavaScript). In any case, we assume that traditional tracking mechanisms are not applicable for this web session, and refer to it as *target* or *private* session.

Additionally, we assume that the victim has another web page opened, containing attacker-controlled JavaScript code. Similarly to the private session, this website can either belong to the attacker, or contain third-party inclusions under attacker's control. However, in this case we assume that the web page *does* have access to information which allows to uniquely identify the user, or store the tracking ID. Therefore, this session is assumed not to be as restrictive as the private session, and is referred to as *receiving* or *non-private* session. More specifically, the private and non-private sessions are considered to be opened:

- in different tabs of the same browser instance (e.g., the user provides login information in only the non-private session, or specifically disallows tracking information for the private session);
- in different instances of the same browser (e.g., the user opens the private session in the private browsing mode, and the non-private session in a "normal" mode).
- in different browsers (e.g., the user opens the private session in the Tor Browser, but has the non-private session opened in another browser for convenience, since the Tor network is usually slower).

Alternatively, instead of having a non-private session, we could assume the attacker to have control over a background application running on the victim's machine. This application does not require any privileged access rights from the operating system, or additional permissions from the user. Therefore, the code can be hidden in any app that already requires Internet access and which the victim is likely to install on his device. This use case complies with existing examples of mobile applications which silently performed user tracking, and have been recently discovered in the wild [Arp+17]. Nevertheless, as this scenario requires an additional application installed, we consider it as less practical, but present evaluation for the sake of completeness. In case the receiver is a web browser, we will refer to this scenario as the *in-browser* scenario; in case the receiver is an additional application, we speak of an *in-app* scenario.

Our general attack scenario is illustrated in Figure 10. As a first step, a victim user visits a target private session **(1)**. For every request, the server generates a unique session ID **(2)** and a specific response for each session ID, and sends it to the client **(3)**. The response is crafted in such a way as to cause the client browser to produce distinct CPU

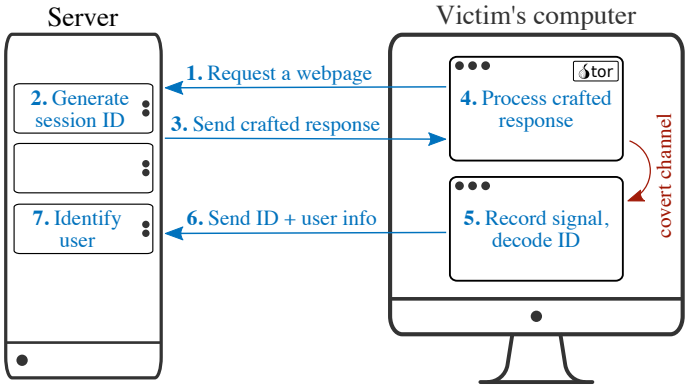


Figure 10: Overview of the general attack scenario utilizing the intra-device CPU-based covert channel.

loads, encoding the ID, when the response is processed **(4)**. We present four different ways that trigger such CPU loads and describe them in detail in the following sections.

Depending on the considered scenario, the information is decoded in different ways:

- In the in-browser scenario, the JavaScript code running in the non-private session estimates CPU loads and decodes the transmitted ID **(5)**. Then the decoded ID is linked to the available non-private session information. Depending on the browser configuration and the method of retrieving load information, this code can be run in background, or only in a foreground tab. We investigate the applicability of each method and browser configuration in Section 3.4.1.
- In the in-app scenario, the attacker-controlled application constantly records the system statistics regarding CPU activity in a background, captures the produced loads and decodes the ID **(5)**. Every ID decoded by the app is considered to belong to a specific user who may be identified by another permanently stored unique token, created for each instance of the recording application.

In both scenarios, the decoded ID is transmitted back to the server together with the aforementioned unique app or browser data **(6)**, which finally allows the server to identify the user by linking the session IDs to the same unique information **(7)**.

3.3 COVERT CHANNEL DESIGN

In this section, we describe implementation details of our covert channel. First, we present our method of encoding data into CPU loads and several approaches in order to produce these loads. Subsequently, we describe and compare three different approaches of measuring these loads with respect to their applicability to in-app and in-browser scenarios. Finally, we examine in detail the decoding process.

3.3.1 *Encoding and transmission*

To encode a binary identifier into CPU loads, we apply on-off keying (OOK) modulation. We force a browser to produce intense CPU activity within a time frame of length t to encode a 1, and perform no activity to encode a 0. Additionally, the transmitted ID is

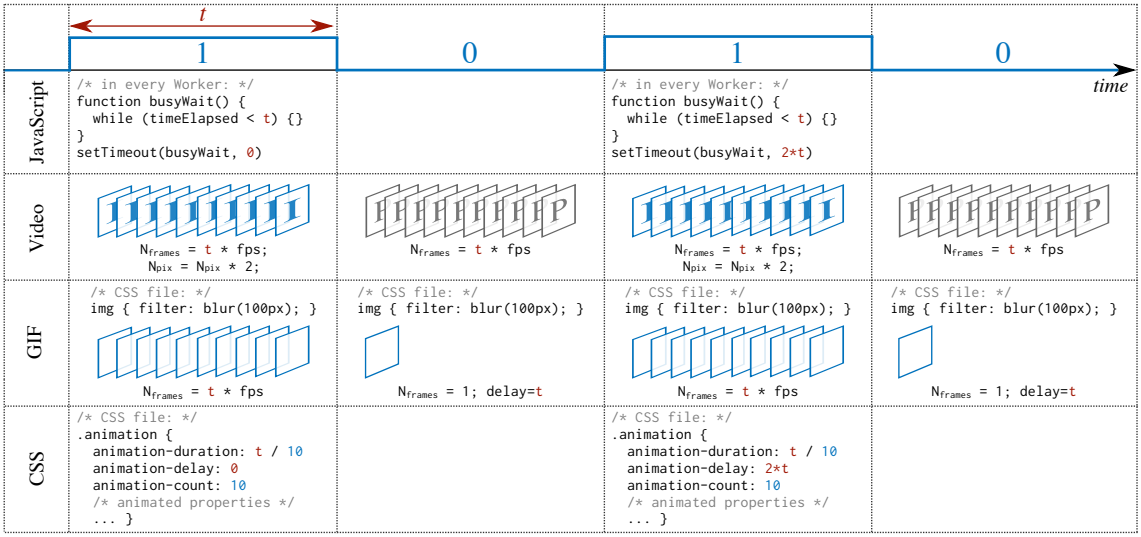


Figure 11: Illustration of four approaches to generate CPU loads: JavaScript, HTML5 video, GIF image, and CSS animations.

prepended with a predefined binary sequence, in order to help the decoding algorithm recognize the start of the transmission. The time periods for transmitting bits of this synchronization sequence and for actual ID bits are of different length, in order to avoid possible false positive detections when the synchronization sequence bits happen to appear in the ID itself. In our implementation, we use the 11-bit Barker sequence for synchronization, due to its low autocorrelation properties [Bar53], which facilitates its detection.

We implement four different methods of producing CPU loads in the browser, using either (1) JavaScript loops, (2) HTML5 videos, (3) filtered GIF images, or (4) CSS animations. Figure 11 illustrates all proposed approaches to encode a binary string into CPU loads, which are explained in more detail in the following subsections.

TRANSMISSION: JAVASCRIPT. Our first approach to generate CPU loads is to execute CPU-intense JavaScript code in the browser. A loop that repeatedly checks if a time period has elapsed (*busy waiting*) results in potential high use of one logical CPU core during this time, corresponding to logical 1. In contrast, for time frames corresponding to the encoding of a logical 0, our implementation forces the target session to *passively* wait, using the `setTimeout` JavaScript function. The relevant JavaScript code, combining time frames of busy and passive waiting, is generated on the server's side for each ID, and then sent to the client for execution.

Furthermore, to increase the CPU load, we utilize the WebWorker API, which allows running scripts in separate background threads. To produce a high CPU load, busy waiting loops are concurrently run in a number of threads, equal to the amount of available logical processors, accessible through the *hardwareConcurrency* property. Therefore, the OS will potentially use all available logical cores to execute the JavaScript loop code. The actual CPU utilization depends on background activities and the OS scheduling mechanism.

Although this approach is an efficient way to utilize up to 100% of CPU time, the necessity of executing JavaScript in the target session makes it potentially less practical, as

JavaScript may be disabled in restrictive browser configurations, e.g., in the Tor Browser. Therefore, we were motivated to find alternative ways to produce intensive CPU activity, which do not rely on the execution of scripts in the target session.

TRANSMISSION: HTML5 VIDEO. The second method we propose in order to create high CPU loads is rendering video with HTML5. Video decoders require more CPU power to process video frames with higher resolution. Moreover, the decoder requires slightly more CPU resources to decompress so-called I-frames, which contain a whole picture, than so-called B- and P-frames, which encode only differences between the preceding and the succeeding I-frames.

Therefore, to increase CPU activity within a time period, we can play a video where each frame has a twice higher resolution, and is encoded as an I-frame (*hard fragments*). In contrast, we can reduce CPU work required for playback within a time frame by playing a video consisting mostly of P-frames (*easy fragments*) with the original resolution. A video combining hard and easy fragments is created as a result, to produce distinct CPU loads.

As the target environment may not allow execution of JavaScript, we may not have the ability to programmatically start and stop video playback in the browser. Instead, the server creates a single video containing a sequence of hard and easy fragments, corresponding to the bits of the ID, to respectively increase or reduce CPU activity. The `autoplay` property of the `video` HTML5 tag containing our crafted video is then set, in order to start playback automatically after loading. Additionally, the `loop` property can be set to continuously repeat the playback in the target session.

The produced video element is set to have a fixed size corresponding to the original resolution. It may be directly shown to the user, as a seemingly benign component of the interface, since the aforementioned encoding process results only in barely noticeable quality differences between hard and easy fragments. Therefore, the encoding process is not evident and the user cannot easily notice that a transmission is taking place. Alternatively, the video object can be hidden from the user (e.g., by setting a negligible opacity, or setting the width and height of a video container to one pixel).

TRANSMISSION: GIF IMAGES. The third approach to produce CPU loads is based on applying CPU-intense CSS styles to GIF animations. The *filter* CSS property allows applying Gaussian blur to web elements. For GIF animations, the filter has to be reapplied for every frame of the image. If the radius of the filter is high, the resulting rendering becomes computationally expensive, even for a very small original image (30x30px in our experiments). Furthermore, the GIF file format allows us to set custom delays after individual frames of the animation. Therefore, a high CPU load can be generated by showing a blurred GIF animation with a maximum available frame-per-second (FPS) rate. In contrast, showing only a single frame with a delay of a time frame duration results in a small CPU load.

The resulting GIF animation can again be hidden by setting a negligible opacity, or be used as a seemingly static benign image, if all the frames of the GIF are identical. The *loop* property can be set to continuously repeat the load generation. GIF animations do not require JavaScript to be shown, and currently cannot be turned off in user-level

settings, even in the Tor Browser, which makes it hard to prevent transmission using this approach.

TRANSMISSION: CSS ANIMATIONS. Our forth approach to produce CPU loads is based on using CSS animations, which make it possible to animate different CSS properties of custom web page elements (e.g., their size, opacity, color, etc.). Declaring CSS animations also do not require JavaScript in order to be declared and executed, and are recommended for usage by modern browser vendors due to their optimized performance [CSS].

Some animated properties, such as scale or rotation, are highly optimized in modern browsers [Ani20], and their execution can be completely moved out of the main GUI thread, or even be done in the GPU. Other properties, however, require recalculation of the page layout or redrawing of elements, and therefore cause intense CPU activity. We select and animate concurrently 10 such properties, including element size, position, color and font, to produce distinct CPU loads. To cause even more CPU loads, we declare these animations for several identical objects at once, and repeat them several times within a time frame. We demonstrate an example of such animation declaration in Appendix A.1.

To transmit each bit set to the logical value 1 in the ID, the server declares a new animation, specifies its `animation-duration` property to the length of a single time frame, and sets its `animation-delay` property to start its playback at the corresponding time frame. Therefore, time frames related to the encoding of bits with logical value 0 result in passive waiting between animations. As one can notice, this approach does not allow us to infinitely repeat the transmission. However, the transmission can be repeated several times by declaring animations for the entire ID several times. Finally, all the animations are made invisible, by setting negligible opacity to all the animated elements.

3.3.2 *Signal retrieval*

The produced CPU loads are to be recorded by either a non-private session or a background application. In both cases, the receiving code constantly evaluates the CPU load at a given frequency. The obtained values are then analyzed by the decoding algorithm. We propose three approaches to record CPU loads: the straightforward method of reading `/proc/stat` data, as well as timing- and sensor-based methods, which indirectly estimate the CPU load. Table 2 provides an overview of implemented approaches.

RECEIVER: `/proc/stat`. If the attacker controls a background application running on a victim's device (the in-app scenario), the system statistics regarding CPU utilization can be directly accessed in order to record CPU loads produced by the target session. On Linux-based platforms, including the Android OS, (up to Android 7), this can be achieved by accessing the `/proc/stat` file, which provides information regarding the amount of ticks (clock cycles) that the CPU has spent performing different kinds of work. By regularly reading the `/proc/stat` file, an estimate of the CPU load can be computed as the ratio of ticks spent on non-idle activities, to all clock ticks within the interval. Therefore, the resulting recorded signal contains discrete-time values ranging from 0 to 1, where 1 represents 100% utilization of CPU.

Table 2: Comparison of methods of measuring the CPU load.

	/proc/stat	Timing		Sensor	
Scenario	app only	app	browser	app	browser
Setup: desktops	+	+	+	–	–
Setup: mobiles	○ (Android⩽7)	+	+	+	+
Background execution	+	+	○ (desktop only)	+	–
Sampling rate	50Hz	40Hz		50Hz	10Hz
CPU usage	⩽ 5%	15–30%		⩽ 5%	

"+": fully supported, "○": partially supported, "-": not supported

We have implemented this method for Linux, MacOS and Android platforms. The recording application can run completely in the background, does not require privileged access or specific permissions from the user, and consumes a small amount of system resources to perform constant recording. Therefore, the recording is hard to detect, and can be a part of a seemingly benign application. The recording sampling rate is limited by the update interval for `/proc/stat` information. In our implementation, we were able to record samples at a frequency of 50Hz, and constant recording required less than additional 5% of available CPU time on all tested devices.

It must be noted that this method is not applicable to the in-browser scenario, as there is no JavaScript API to directly access system statistics of CPU utilizations, or to access the file system. Moreover, starting from Android 8 the `/proc/stat` is no longer accessible from mobile applications. Nevertheless, we present evaluation results for this method, to be able to compare indirect ways to estimate CPU activity with precise system information provided in `/proc/stat`.

TIMING-BASED RECEIVER. In order to estimate the CPU activity, the receiving browsing session or app constantly executes a small code segment and measures its execution time. The operating system has to allocate CPU resources to the target session when its execution requires intense CPU activity, and therefore the execution of the receiving session code takes more time. Therefore, we can use the execution time of a code segment run in the receiving session as estimate of the overall system CPU utilization within the recorded interval. For this reason, the resulting recorded signal consists of discrete-time values of the execution times of the code fragment.

This approach can be run from a background application, as well as from a background non-private tab having the receiving session opened. On mobile devices, browsers are prevented from executing JavaScript in background tabs, in order to reduce power consumption. Therefore, on mobile devices the in-browser scenario is limited to the use case when both private and non-private sessions are opened side by side in the foreground.

The recording code in the receiving session has to consume a significant amount of CPU time, in order to create a competition for CPU resources between the two sessions. Moreover, in order to capture the CPU activity performed by the target session, the recording code needs to be run on several available physical threads. In our implementation, we execute a recording with a sampling rate of up to 40Hz, running counting loops

in multiple threads using the Web Worker API, consuming 15–30% of the overall CPU time on tested devices.

The optimal amount of threads to be used, as well as the number of iterations to be executed in these threads, depends on a particular hardware configuration. Therefore, to choose the best parameters, the receiving session can perform a calibration phase right after being opened. More specifically, the receiving session can initiate a transmission of a predefined sequence using the JavaScript-based method described above in Section 3.3.1, and at the same time execute the receiving code fragment with different parameters, measuring the resulting execution time difference between intervals corresponding to bits 1 and 0. The number of threads and iterations which results in a higher difference, is then used for continuous decoding. In our implementation, we probe a set of 16 different parameters, and the resulting calibration phase can be finished within 8 seconds.

Interestingly, for some configurations we observed the opposite effect: the execution time of the receiving session becomes noticeably *smaller* when CPU-intensive code is executed by the target session. This effect occurs due to CPU throttling: when applications require only a small amount of CPU resources, the system throttles the performance to save power; when the target session triggers an intense CPU load, the system increases its performance, and the code in the receiving session executes faster. Our decoding implementation allows us to address this case by additionally checking the cross-correlation with the inverse synchronization sequence.

SENSOR-BASED RECEIVER. Finally, we propose a sensor-based approach to receive the covert signal on mobile devices, based on the reaction of magnetometers to the electromagnetic interference. As we discussed in the previous chapter, magnetometers can capture EM signals emitted by a nearby laptop's CPU or hard drive. In this work, we discover that the magnetometer on a smartphone can be affected even by the EM signal emitted by its own CPU. Moreover, the higher the CPU activity is, the stronger the relevant EM emissions are, and, therefore, also the higher the resulting disturbance is. Thus, by constantly observing the magnetometer data, we can indirectly estimate the produced CPU loads.

Sensor data can be accessed in Android applications by using the Sensor API. Moreover, a new Generic Sensor API [WPS18], introduced in Google Chrome 63 in 2017[1], allows accessing magnetometer data from web pages. Therefore, this method is applicable to both in-app and in-browser scenarios. In order to convert the three-dimensional magnetometer data into discrete-time values, we follow the solution proposed in Chapter 2, by applying Principal Component Analysis [Hot33] to the data and choosing the first component as the result, as it represents the direction with the biggest data disturbance.

Although the Generic Sensor API allows accessing sensor data only from foreground tabs, and limits the sampling rate to 10Hz, this method provides a good estimate of the CPU load without consuming many resources, unlike the timing-based approach. In our in-browser and in-app implementations, we were able to record magnetometer data using less than 5% of available CPU time.

1 At the time of writing this dissertation, the Generic Sensor API has the status of the W3C Candidate Recommendation; access to the magnetometer sensor requires a developer flag to be enabled in the browser settings.

Table 3: Applicability of the four approaches proposed to generate CPU loads.

	JavaScript	HTML5 video	GIF	CSS animations
Browser applicability	all tested browsers			
Background tab execution	fully supported [a]	fully supported	not supported	not supported
Tor Browser security level that permits execution	low (default) level	low (default) level [b]	all levels	all levels
Prevention ways in browsers	disable JavaScript	disable automatic playback	impossible without plugins	impossible without plugins
Ways of detection	CPU monitoring, JavaScript code analysis	CPU & network monitoring, HTML5 video file analysis	CPU monitoring, CSS and GIF file analysis	CPU monitoring, CSS source analysis
Traffic overhead	<1kB	≈3Mb for 30s video	10–100kB	<2kB
Transmission speed (with BER ≈ 10%)	*/proc/stat*: 20–30bit/s timing: 10–20bit/s sensor: 8–12bit/s	*/proc/stat*: 3–4bit/s timing: 1–3bit/s sensor: 1–3bit/s	*/proc/stat*: 7–12bit/s timing: 5–8bit/s sensor: 5–8bit/s	*/proc/stat*: 13–15bit/s timing: 8–12bit/s sensor: 5–8bit/s

[a] JavaScript can be throttled or even blocked in background tabs in mobile browsers

[b] On medium and high security levels, video playback can be triggered by the user.

3.3.3 *Decoding*

The decoding process is identical for three recording scenarios. First, we calculate the cross-correlation between the recorded signal and the predetermined synchronization sequence, both resampled to have the same sampling rate. A time point corresponding to a high peak in the cross-correlation is considered as the start of a transmission.

We finally start decoding the recorded signal from the point where it matches the predetermined synchronization sequence. Then, using the synchronization sequence, we calculate the average value of all measurements within time frames corresponding to a logical 0, and, similarly, the average value of all measurements corresponding to a logical 1. Subsequently, we define the mean of these two averages as a threshold. Afterwards, the transmitted ID is decoded bit by bit, by comparing this threshold with the average of measurements within a time frame corresponding to a bit of the ID. If this average is above the threshold, the bit is decoded as 1; otherwise, the bit is decoded as 0.

3.4 EVALUATION

In this section, we evaluate the presented covert channel. First, we compare the applicability of the four proposed methods of CPU load generation to different browser setups. Then, we evaluate each solution on different hardware and software configurations, by measuring the signal-to-noise ratio (SNR). Afterwards, we determine the achieved transmission speed for all proposed methods of transmission and reception of the signal. Finally, we evaluate the robustness of the signal in the presence of background CPU noise.

3.4.1 *Applicability*

In this section, we investigate browser configurations applicable for each of the four approaches proposed to generate CPU loads, specifically focusing on their applicability to the Tor Browser. Additionally, we analyze available ways in browsers to prevent the success of these methods, described how the transmission can be detected, and measured the traffic overhead introduced by each approach. Table 3 summarizes the results. For convenience, we also include in the table the average achieved bitrates for all three recording approaches, discussed in more detail in Section 3.4.3.

We confirm that all four approaches work in popular desktop and mobile browsers. The default configurations of all tested browsers[2] allow execution of JavaScript, as well as Web Workers API support, playback of HTML5 videos, showing filtered GIF animations, and execution of CSS animations. To prevent successful generation and transmission of the signal, some settings need to be changed. In particular, JavaScript and automatic video playback can be manually disabled in browsers. However, as mentioned before, the generated video can be presented to the user as a seemingly benign component of the interface. In the Tor browser, access to sensitive content is regulated through the so-called "Security slider" with three security levels, which represent a trade-off between usability and security, by disabling several components at each level [Per+20]. All four approaches to create CPU loads work at a default (Low) security level. At the Medium and High levels, untrusted JavaScript and video autoplay are disabled. However, the GIF and CSS animations approaches work even at the highest security level.

Furthermore, GIF and CSS animations cannot be disabled at all using the settings of all tested browsers, including the Tor Browser. To prevent the transmission, users have to install an extension which allows blocking GIF animations or overriding rules declaring CSS animations. The potential shortcomings of the GIF and CSS animations approaches are that a target web page has to be opened in a foreground tab and the browser window must not be minimized. In contrast, JavaScript-based and video-based generation methods work even in minimized browsers and background tabs.

A straightforward way to detect covert transmission in all four cases is to monitor the CPU activity. Additionally, a user can manually observe the sources of the web page and analyze the executed JavaScript code, or the declaration of CSS animations, or discover the hidden video or GIF element. The video approach also introduces a traffic overhead which may be noticed at the network level. However, even if the components of the attack are discovered, their purpose may remain unclear to the user. In particular, additional file content analysis is required to recognize hard and easy fragments in the video file, or a significantly different amount of frames in the GIF animation. Moreover, both JavaScript code and CSS animation declarations can be obfuscated to hamper the analysis.

In summary, we believe that all four approaches are applicable to modern desktop and mobile browsers. The methods based on GIF and CSS animations introduce a significant privacy risk for users, since they work even at the highest security level of the Tor Browser, and cannot be disabled in browser settings.

2 Tested browsers: Chrome 63.0 (desktop and mobile versions), Firefox 57.0 (desktop and mobile versions), Tor Browser 7.0.

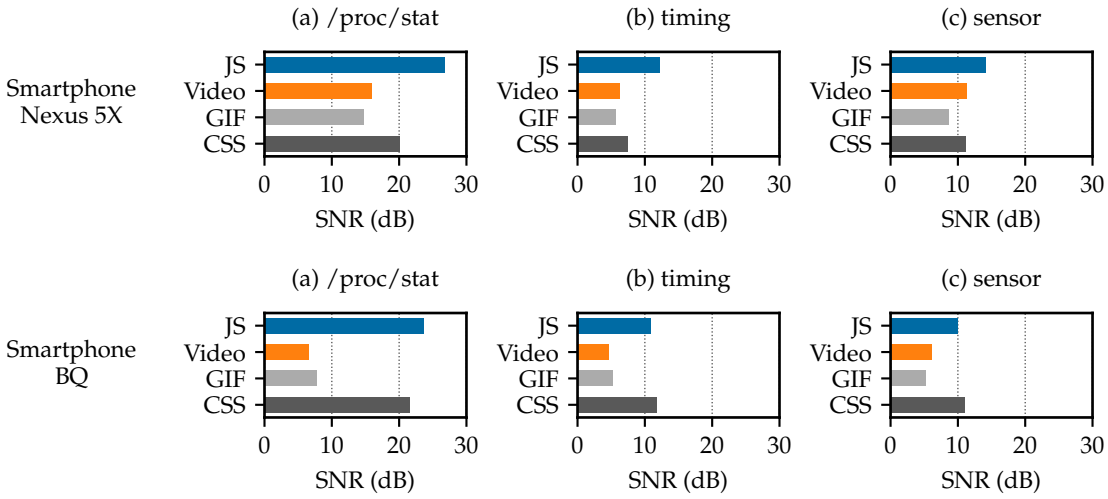


Figure 12: SNR levels recorded for two smartphones for four proposed methods of CPU load generation, received using `/proc/stat` (a), timing (b), and sensor (c) approaches.

3.4.2 *Signal strength*

In this section, we measure and compare the strength of the signal for each of the transmitting and recording methods described in Section 3.3. For this purpose, we measure the signal-to-noise ratio (SNR) of the produced signals, i.e., the ratio of the difference between the two signal levels (corresponding to high and low CPU loads), to the standard deviation of measurements at the base CPU load level, when no loads are induced by the target session. For these measurements, we alternate high and low CPU loads lasting for 2 seconds, and repeat this pattern 50 times. In this experiment, we use two smartphones: a Nexus 5X and a BQ Aquaris X5, both running Android 7.1[3].

Figure 12 demonstrates the SNR measured on a Nexus 5X and on a BQ Aquaris smartphones, recorded by accessing `/proc/stat` **(a)**, measuring timing response **(b)**, or analyzing magnetometer disturbances **(c)**. A separate Google Chrome instance process is used for recording using timing and sensor-based approaches. We have checked that timing recordings made in Mozilla Firefox on a laptop produce similar results with slightly lower SNR (≈20%) due to a higher "noise" level. We have also confirmed that recording magnetometer data using the native application results in the same SNR (within ≈20%).

One can notice that all four approaches generate a distinct signal on both tested devices. The JavaScript method directly occupies all available CPU cores, which results in the highest SNR in all recording scenarios. GIF animations, CSS animations and HTML5 videos request only a part of the available CPU resources, and therefore, the signal is lower in comparison to JavaScript-based generation. Although the SNR levels observed for the video-based method appear to be comparably low, we are able to distinguish the signal levels and correctly decode transmitted IDs, even for the minimum SNR value of ≈4 dB observed for this method.

One can also notice that the timing-based approach, which relies on competition between execution threads, results in a lower SNR for non-JavaScript approaches than

3 In the corresponding publication [Mat+18], we also provide evaluation results for the desktop platform.

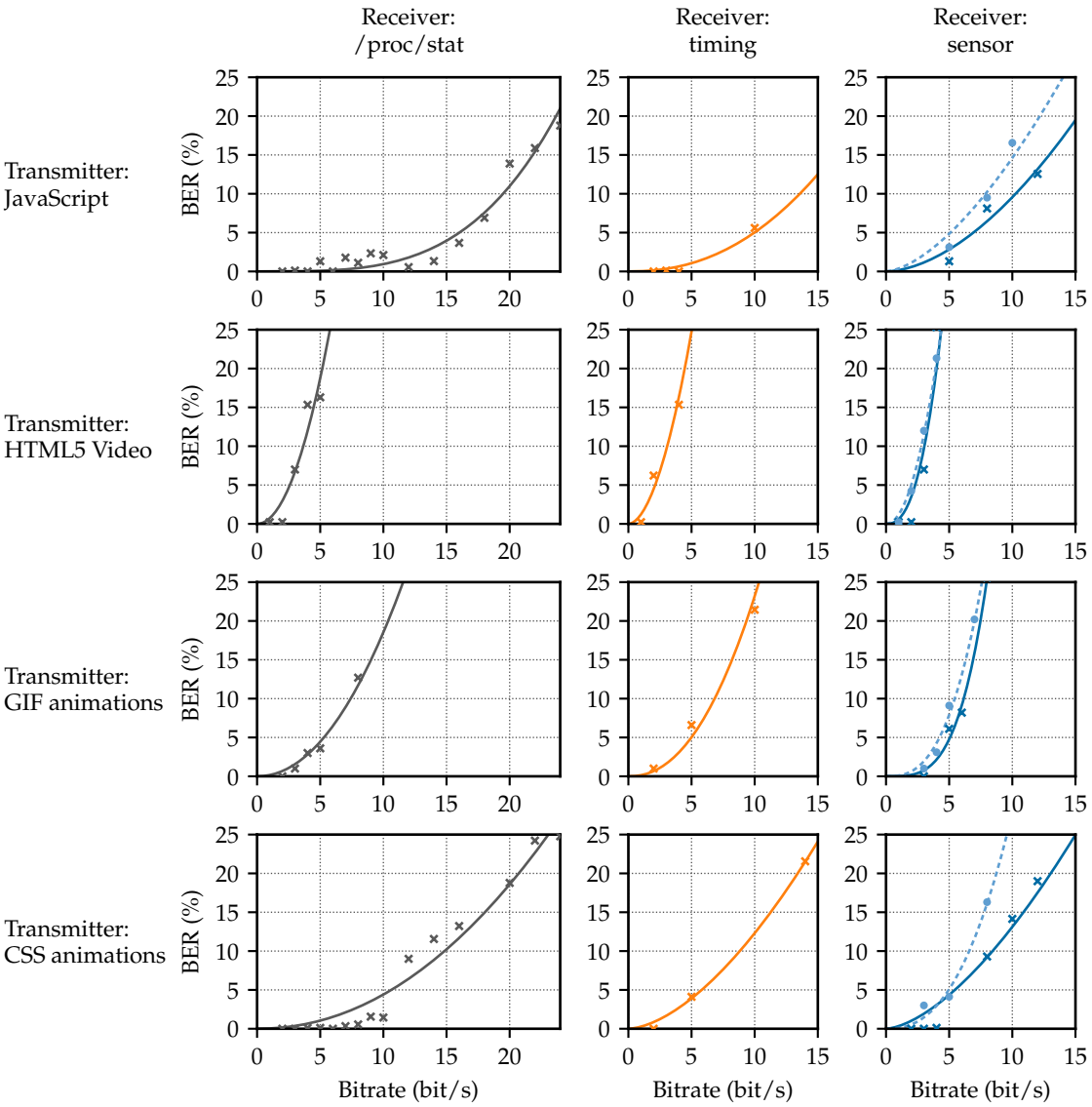


Figure 13: Bit Error Rate (BER) corresponding to different bitrate for each encoding method on a smartphone using the `/proc/stat` receiver (first column), the timing-based receiver (second column), and the in-app (solid) and in-browser (dashed) sensor-based receivers (third column).

`/proc/stat` approach, which has access to cumulative CPU activity across all threads. Finally, one can notice that sensor-based approach generates a distinguishable signal on both smartphones, although it requires much less resources than the timing approach.

As a result, we consider all the approaches discussed as practical and potentially capable of successfully transmitting and receiving a signal under different hardware and software configurations.

3.4.3 *Transmission bitrate*

In this section, we evaluate the optimal speed of covert transmission using each generation method and three proposed receivers, when using the on-off modulation scheme described in Section 3.3.1.

For this purpose, we transmit IDs through the covert channel with different bitrates, and calculate the decoding bit error rate (BER). We consider a BER of 10% as practically suitable, since in this case correct decoding can be achieved by using error-correcting codes. For each bitrate we transmitted 50 IDs of a length of 30 bits, each prepended with an 11-bit synchronization sequence, which potentially allows one to identify more than 1 billion unique sessions. We choose a smartphone Nexus 5X for this experiment, using Google Chrome to open the target session.

Figure 13 demonstrates the dependency between BER and bitrate for all four methods of generating CPU loads, using the `/proc/stat`-based receiver (**1,3**) and the timing-based receiver, as well as using the sensor-based receiver in both in-app and in-browser implementations.

One can see that the JavaScript method achieves the best transmission speed, with up to 20 bit/s with <10% BER on a smartphone when using the `/proc/stat`-based receiver. Since only the JavaScript approach allows us to directly request all available CPU resources with precise timing, bitrates achieved for signals produced with CSS animations and GIF animations are comparably lower. Still, for CSS animations we achieve bitrates of 5–15bit/s, depending on the receiver. The signal produced by GIF animations is less stable on a smartphone, apparently due to a different implementation of blur filtering in mobile browsers. Finally, the lower SNR provided by the video-based method results in a noisy signal even at a comparably low transmission speed of 1–5 bit/s. Therefore, we consider this approach suitable only for short IDs, e.g., with length of 10 bits.

When using the timing-based approach, more errors appear at high bitrates for all signal generation methods, since the used decoding method relies on a less precise CPU load estimation mechanism. Similarly, the sensor-based receiver on the smartphone provides less precise and more noisy signal, but still achieves successful decoding at 5–8bit/s for transmission using CSS and GIF animations. The in-browser implementation of the sensor-based receiver (with a sampling rate limited to only 10Hz) remains stable for low speed, with additional errors appearing at high bitrates.

We can conclude that all proposed methods allow to covertly transmit identifiers within several seconds, and reliably receive them by reading the `/proc/stat` file, as well as indirectly estimating the CPU activity using the timing- and sensor-based approaches.

3.4.4 *Robustness*

In this section, we evaluate our covert channel in the presence of background CPU activity. For this purpose, we run transmissions on a laptop simultaneously with synthetic stress tests using the `stress-ng` utility, involving 5, 10 and 20% of CPU time and utilizing all available threads, and measure the resulting BER. Furthermore, as a real-life workload scenario, we also evaluate transmission in presence of videos played in background on the laptop and on the smartphone, consuming 6–22% of CPU time. Based on the previous

Table 4: Robustness of the transmission: BER under different background activity.

	/proc/stat		Timing		Sensor	
	5bit/s	8bit/s	5bit/s	8bit/s	5bit/s	8bit/s
Laptop						
no noise	1%	2%	2%	4%	–	–
stress 5%	6%	9%	10%	12%	–	–
stress 10%	8%	11%	22%	26%	–	–
stress 20%	16%	23%	26%	31%	–	–
video 6%	1%	3%	8%	10%	–	–
video 22%	13%	14%	33%	36%	–	–
Smartphone						
no noise	1%	3%	4%	9%	3%	8%
video 6%	2%	6%	9%	15%	4%	10%
video 20%	8%	17%	22%	28%	8%	18%

experiment, we choose CSS animations as the signal generation method, as it performs better among other methods, not utilizing JavaScript. Table 4 shows the resulting BERs for two fixed bitrates, using all three recording methods.

As one can see, the higher background activity, the more decoding errors appear, as any peak activity can be misinterpreted as transmission of bit 1 with the on-off modulation scheme applied. For this reason, video playback causes fewer errors than artificial stress activity (consuming similar total CPU time), as it results in "smoother" use of CPU. The `/proc/stat` receiver, expectedly the most robust one, provides successful decoding with up to 10% stress noise. The timing-based approach is the most susceptible to background CPU activity, as additional running processes cause more system interruptions and context switches, which affect the used CPU estimation. The sensor-based approach, however, is comparably stable under low noise from the video playback, especially at low bitrates. Nevertheless, all the recording methods allow us to successfully receive the signal under low CPU activity caused by playing videos in background, or stress activity under 5%.

As a result, we believe that all the approaches are applicable to typical usage scenarios (web browsing, video playback) with low background activity; under high noise, the decoding becomes unstable, and more robust encoding schemes may be necessary to improve quality of transmission.

3.5 COUNTERMEASURES

A number of potential countermeasures exist against the web-tracking methods described in this work. Nevertheless, the majority of these countermeasures would require modifications in the software or hardware of the target device, and may have significant performance drawbacks.

The most straightforward way to prevent the attack of the in-browser scenario has been proposed in [Whi20]: limit the amount of available CPU resources for all browsers, e.g., by using the *cgroups* kernel feature on Linux platforms. For example, if each of the two

browsers employed in the in-browser scenario is allowed to only use 25% of the CPU, the CPU loads induced by our code in the target session will not affect the execution time of the code in the receiving session. However, this countermeasure significantly reduces the performance of web browsers in general, and strongly contradicts the industry trend to increase the complexity of web applications. Moreover, correct decoding of the ID may still be possible if the code within the receiving session runs faster than usual when the target session's code is also running on the CPU, due to the throttling effects that we have described earlier in Section 3.3.2.

Another countermeasure in order to hinder usage of the CPU-based covert channel could be to randomize CPU activity, in order to balance and normalize the loads produced, and thus prevent the recognition of either the synchronization code preceding the identifier or even the correct decoding of the CPU loads to bits. However, in such a case, all applications would have to endure excessive overheads in their processing times and thus the performance of the overall system would also be significantly impeded.

Preventing the execution of JavaScript in background tabs on all platforms would limit the applicability of the covert channel, as both our JavaScript-based transmitter and the timing-based recorder would only work in foreground tabs. Currently, the background execution of JavaScript is limited in mobile browsers, and throttling of JavaScript events in background tabs is recently introduced for the desktop Chrome browser [Chr20]. However, these limitations are not applied to code executed in separate threads using Web Worker API (as in our approach). The Chrome development team considers restricting background Web Worker execution at some time in 2018 [Int20], and we believe that similar measures must be taken for modern browsers. Nevertheless, the described HTML5 video approach would not be affected by this change, while GIF and CSS animations are already not executed in background tabs.

Additionally, obtaining precise timing information via the Performance API in JavaScript can be forbidden to limit the effectiveness of the in-browser decoding, at least for background tabs and in restrictive browser configurations, such as Tor Browser. However, we believe that decoding will still be possible with lower bitrates.

Unless CSS animations and GIF animations are disabled by default in the most restrictive mode of the Tor Browser, the related attack scenario is very difficult to prevent. The Tor browser development team already considered this idea to prevent timing information from being indirectly available through CSS Animations [Tor20], and we strongly advocate this measure.

Finally, to prevent the direct way of accessing CPU statistics, access to the `/proc/stat` can also be disallowed for user-level applications on all platforms, as has been made for Android 8. Furthermore, even allowing access to the `/proc/stat` file only at a reduced frequency could potentially make it more difficult to measure CPU load correctly and thus hinder successful identification of the CPU-based transmission. Nevertheless, in this work we show that both timing- and sensor-based techniques can be used instead to indirectly estimate the CPU load.

Physical isolation (shielding) of the smartphone magnetometer from the CPU can mitigate the sensor-based recording. However, it would require hardware changes, and contradicts to the industry trend of making mobile devices thin and compact. We believe that the sampling rate of accessing magnetometers can be further reduced, although, as has been shown, even a 10Hz rate in the in-browser scenario is enough to successfully

transmit the tracking ID. Furthermore, access to magnetometers can be restricted in background applications (as has been introduced in Android 9), and require and explicit user permission.

3.6 SUMMARY

In this chapter, we presented the use of covert channels for the purpose of web user tracking. We showed that different web components, such as HTML5 videos, GIF animations, or CSS animations, can produce distinct CPU loads when executed in the browser. The transmission can be initialized without any client-side code, which makes the solution applicable to very restrictive browser configurations. An identification token encoded into the CPU loads can be effectively exfiltrated from a private browsing session to either another session, or to an app recording in the background. To capture the produced loads, the receiver can observe system statistics about CPU activity, or measure execution timing for a known segment of code. Furthermore, we demonstrate how susceptibility of magnetometers to electromagnetic activity caused by smartphone's CPU can effectively be used to estimate CPU activity. Although the resulting bit error rate becomes too high in presence of background noise, sensor-based approach allows us to reliably receive the covert signal under low background activity. We therefore conclude that the introduced intra-device covert channel is applicable to the web-tracking scenario, and this way poses a significant threat against the privacy of online users, even for those who use more restrictive web browsers, such as the Tor Browser.

4 INTRA-DEVICE MAGNETIC SIDE CHANNEL

In this chapter, we further analyze the susceptibility of magnetometer sensors to the CPU activity of a smartphone. We investigate how the sensor's reaction can be exploited to establish a passive side-channel attack that infers running applications and websites, by analyzing the disturbance of the magnetometer caused by the corresponding CPU activity patterns. We evaluate the sensor's reaction on a large number of smartphones confirming the overall presence of the cross-component interference, and discuss the applicability of a malware to covertly record magnetometer values on modern versions of mobile operating systems and in web browsers.

Remarks. Content in this chapter is based on the corresponding publication [Mat+19].

4.1 MOTIVATION AND CONTRIBUTIONS

Mobile devices have become ubiquitous in people's daily activities. According to recent studies, adults spend more than 2.5 hours per day on their smartphones or tablets [Lu18], the average user runs over 30 mobile applications per month [Retr18], while mobile Internet traffic already exceeded desktop usage [Eng18]. Extensive mobile usage results in an increasing amount of personal information that is stored and processed on mobile devices, which increases risks of its unauthorized or malicious misuse. Fortunately, mobile operating system developers put a great deal of effort to limit such risks, by isolating running applications into sandboxed environments and by introducing permission-based access restrictions for sensitive components [App18c; App18a].

Nevertheless, several previous studies have shown that an attacker can exploit side-channel leakage to infer information about applications and websites opened on a victim's mobile device (referred to as application and website fingerprinting). These leakage sources include network traffic statistics [Spr+16; Zha+18; Zho+13], power consumption traces [Che+17; Yan+15], CPU utilization [ZW09; SXA16], memory usage statistics [JS12; CQM14; Zha+18], and other information available through the *procfs* pseudo file system [Spr+18] or system APIs [SPM18; Zha+18]. The information obtained from application and website fingerprinting can potentially reveal sensitive information about the user, e.g., hobbies, political interests, religious beliefs, or health conditions. The more actively a victim uses the device, the more precise is the resulting user profile.

To prevent such attack vectors, operating system developers have gradually restricted access to system resources which can reveal sensitive information. In particular, starting from Android 7, applications cannot access pseudofiles revealing system information about other processes (e.g., */proc/[PID]*) or monitor traffic statistics of other applications [And18b]. Similar access restrictions to per-process statistics are applied to applications on iOS devices starting from iOS 9 [Zha+18]. Furthermore, starting from Android 8 and on the most recent Android 9, the access to global system statistics available through *procfs* and *sysfs* is restricted [Goo18], preventing application and website fingerprinting attacks based on CPU utilization and power consumption traces.

In this chapter, we demonstrate that reaction of magnetometers to CPU activity of a smartphone can provide an alternative source of side-channel leakage for website and application fingerprinting. More specifically, we propose to use this effect as a passive side-channel attack which aims to identify running activities. We show that magnetometer disturbance patterns closely represent CPU workload. Therefore, they allow attackers to fingerprint browsing and application activity with an accuracy comparable to the method based on observing overall CPU statistics available through *procfs* before Android 8. The proposed method does not require any user permissions at the moment. As a result, any application installed on a device can infer running applications or visited websites, unnoticeable to the end user. Furthermore, as we mentioned in Chapter 3, the magnetometer can now be accessed within web pages using the Generic Sensor API [WPS18]. In this case, the attack does not even require an installed malicious application. Instead, a web page under the attacker's control can establish fingerprinting of other web pages or applications.

We have examined 80 popular smartphones and tablets, and have found that magnetometers on 56 of them are affected by CPU activity. For these devices, we created a classifier which analyzes disturbances in recorded sensor measurements to identify browsing and application activities on a device. In practical scenarios, we are able to correctly identify an opened website with an accuracy of up to 91% for a set of 50 popular websites. We were also able to identify a running application with up to 90% accuracy for a set of 65 candidate applications. In all cases, the accuracy is significantly higher than the baseline accuracy obtained from random guessing, and is comparable to the approach based on analyzing *procfs* information. Therefore, the presented sensor-based side channel can pose significant privacy risks to end users.

CONTRIBUTIONS. Our contributions can be summarized as follows:

- We further investigate the reaction of magnetic sensors to varying CPU activity on 80 different smartphones and tablets in cloud and lab environments. To the best of our knowledge, our work is the first to test this side channel on a large number of Android and iOS devices.
- We propose to exploit this side channel for application and website fingerprinting on mobile devices. We show how to extract information from magnetometer disturbances, evaluate the classification performance under realistic conditions, and discuss possible countermeasures.
- We show that our method provides classification accuracy comparable to techniques based on *procfs* leakage, but works in presence of security enhancements implemented in the latest mobile operating systems, and can be run in both in-app (malicious app) and in-browser (malicious web page) scenarios.

4.2 BACKGROUND

4.2.1 *Sensitivity of magnetometers to CPU activity as a side channel*

In this chapter, we further investigate the reaction of magnetometers to CPU activity on mobile devices. We have observed that on many smartphones the pattern of the

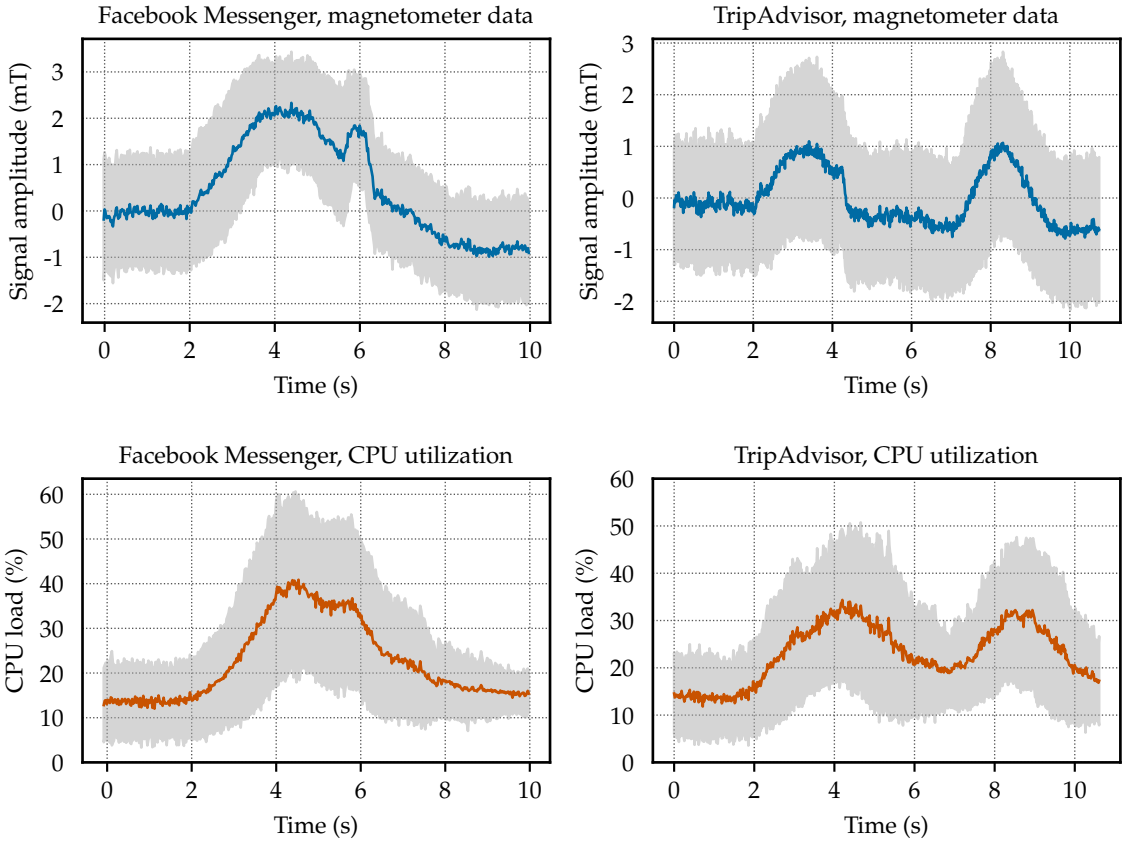


Figure 14: Examples of CPU utilization and magnetometer measurements, recorded during opening two applications on a Google Pixel 2 smartphone. Plots represent mean and standard deviation for 175 samples. The CPU and sensor data are visually correlated with each other for each application and are significantly different between applications.

sensor disturbance accurately follows the pattern of the CPU utilization: the higher the CPU load, the higher is the resulting peak in magnetometer measurements, and vice versa. The reasons for this are the following: On one hand, the CPU is one of the most power-consuming components of the device [CH10]. The screen and GSM module can consume more power, but their consumption remains comparably stable during normal usage. On the other hand, mobile processors are optimized to consume minimum power under low or idle activity.

At the same time, different applications or websites require different amounts of CPU resources when running. As a result, CPU utilization traces, as well as the corresponding interference observed in sensor measurements, can contain distinct patterns which uniquely identify the activity. Figure 14 shows CPU utilization traces recorded on a smartphone for two applications, in combination with magnetometer readings recorded at the same time. The patterns in the corresponding CPU and sensor measurements are visually correlated for each application, they are stable within multiple recordings, but distinct for two different applications. In this work, we show that an adversary can effectively extract information out of such recorded magnetometer measurements, and use it to perform application and website fingerprinting on a victim's device.

Table 5: Comparison with other related works exploiting side-channel information leakage for website and/or application fingerprinting.

Work	Attack[a]	Leakage source	Platform	Blocked
Jana & Shmatikov [JS12]	WF	memory footprint	mobile apps	Android 7
Zhou et al [Zho+13]	AF	data-usage statistics	mobile apps	Android 7
Spreitzer et al. [Spr+16]	WF	data-usage statistics	mobile apps	Android 7
Gulmezoglu et al. [Gül+17]	WF	hardware perf. events	desktop apps	Android 7
Chen et al. [Che+17]	AF	power traces (SW)	mobile apps	Android 8
Clark et al. [Cla+13]	WF	power traces (HW)	desktop apps	Android 8[b]
Yang et al. [Yan+17]	WF	power traces (HW)	mobile apps	Android 8[b]
Diao et al. [Dia+16]	AF	system interrupts	mobile apps	Android 8
Spreitzer et al. [SPM18]	AF&WF	several Android APIs	mobile apps	not blocked
Spreitzer et al. [Spr+18]	AF&WF	several *procfs* resources	mobile apps	not blocked
Shusterman et al. [Shu+19]	WF	cache occupancy	desktop browsers[c]	not blocked[c]
Our work	AF&WF	magnetometer data	mobile apps and browsers	not blocked

[a] WF — website fingerprinting; AF — application fingerprinting
[b] Attacks use power traces collected using hardware; prevention is specified for *sysfs* traces
[c] Evaluation is presented for desktop browsers, but potentially generalizes for mobile platforms

4.2.2 *Related work*

Researchers have shown that different side-channel information can be used to infer applications and websites opened on a smartphone. Jana and Shmatikov [JS12] observed the memory footprint of a browser (available through *procfs*) to enable website fingerprinting. Zhou et al. [Zho+13] and Spreitzer et al. [Spr+16] showed that the Android data-usage statistics API provides precise information about network activity and allows fingerprinting applications and websites. Gulmezoglu et al. [Gül+17] used information about system performance counters to establish website fingerprinting, whereas Diao et al. [Dia+16] exploited information about system interrupts to establish application fingerprinting, with both leakage sources available through *procfs*. Several researchers showed that power consumption traces, collected through *sysfs* [Che+17; Yan+15], using a malicious charger [Yan+15] or a malicious battery [Lif+18], are highly correlated with the CPU activity pattern, and therefore, also can be used as leakage source to infer opened applications [Che+17; Yan+15] and websites [Cla+13; Yan+17; Lif+18]. Recently, Spreitzer et al. discovered multiple leakage sources available through *procfs* [Spr+18] and Android APIs [SPM18], which allow inferring website and application activity. Finally, several works have been presented on microarchitectural side channels, which can be used to infer information about visited websites [Ore+15; Lip+18]. In the most recent work, Shusterman et al. [Shu+19] demonstrated the cache occupancy side channel to establish website fingerprinting in the in-browser scenario. Although the results were presented for the desktop platform, the approach may be applied to mobile devices.

Table 5 summarizes these prior works and compares them with our approach. As we can see, most of the leakage sources are already blocked in the latest Android OS. Furthermore, currently available *procfs* resources can be blocked in future versions of Android without serious impact on existing applications as they provide system-specific technical information. In contrast, our attack works on the latest Android 9 and access to magnetometer cannot be completely blocked, since numerous applications rely on magnetometer values (e.g., navigation applications). Furthermore, almost all prior works require a malicious application to be installed on a device, while our attack can be launched from a web page.

In a recent work [Che+19], Cheng et al. exploited the reaction of magnetometers to EM activity to infer applications and webpages opened on victim's laptop located in vicinity to the attacker's smartphone. In this work, we show that magnetometer disturbance on mobile devices accurately represents the patterns of the *internal* CPU activity, evaluate this effect on a large number of modern devices, and show that a malevolent application on a victim's smartphone can infer running activity, namely, to perform application and website fingerprinting.

4.3 ATTACK SCENARIO: APPLICATION AND WEBSITE FINGERPRINTING

In this section, we discuss two considered attack scenarios, *in-app* and *in-browser*, discuss their limitations and elaborate on the considered assumptions.

In the *in-app* scenario, the victim installs an attacker-controlled application on his or her device. This application does not require privileged access rights from the system and does not have additional user permissions, apart from access to the Internet, granted by default. Therefore, malicious code can be hidden in any application which victims are likely to install. This application can be sandboxed according to the latest Android and iOS security enhancements. In particular, this application does not have any information about other running applications or network traffic, and does not have access to system resources (e.g., over *procfs* or *sysfs*). The attacker only has access to zero-permission sensor information.

In the *in-browser* scenario, a victim opens a web page under the attacker's control. The web page either fully belongs to the attacker, or contains components from an attacker-controlled server, similarly to the case when websites include third-party code from advertisement and analytics services. Such third-party components can be present on thousands of websites, which makes this scenario comparably even more scalable. Similarly, we assume that this web page is sandboxed by the browser from other web pages, processes, and system resources.

In both scenarios, an attacker constantly collects magnetometer readings and tries to identify opened applications or websites by applying a supervised learning approach. To achieve this, the attacker needs to perform a training phase, which requires gathering a sufficient set of labeled traces for each visited website or application. A powerful attacker can perform learning on a large number of devices which he or she owns or accesses using cloud testing platforms, such as AWS Device Farm [AWS20]. On a victim's device, the attacker only collects traces to be classified during the testing phase and sends them to a server. The attacker may additionally send information about the victim's device to a server, to match the victim's device with same model of the device that the attacker

Table 6: Comparison between the in-app and in-browser attack scenarios

	In-app scenario	In-browser scenario
Recorder	native app	web page
Sensor access	Sensor or Core Motion frameworks	Generic Sensor API
Sampling rate	50–100Hz	10Hz
Background recording	Android $\leqslant$ 7: full Android 8–9: partial (or full with notification) iOS 5 and newer: partial	n/a
Attack code distribution	Application markets	Phishing web links or 3rd-party JavaScript inclusions
Scalability	medium	high

trained on, as model-specific classification has a higher success rate (we evaluate this in Section 4.5.2). On Android, the device model is freely accessible by applications through the *Build.MODEL* property; on iOS, it can be retrieved using the *UIKit* framework and *uname* system call. In the in-browser scenario, this device model can be obtained from the User-Agent HTTP header [Use20].

Alternatively, for the website fingerprinting case, an attacker can perform the learning phase directly on the victim's device. For this purpose, a malicious application can embed an invisible WebView [Web18] component to open all websites from the training dataset and send the labeled sensor data to an attacker-controlled server. Although such an approach would provide the most precise device-specific training dataset, in this case, the application needs to be actively used in the foreground by the victim for a significant amount of time.

4.3.1 *Applicability of the scenarios*

The in-app and in-browser attack scenarios also differ regarding their applicability. Due to technical limitations of the Generic Sensor API, the magnetometer can only be accessed from foreground browser tabs. Therefore, the in-browser scenario can only be used to identify either *background* activities, or websites and applications opened side by side with the recording web page, in so-called *split screen* mode.

In the in-app scenario, the time frame during which the malicious application can gather magnetometer traces depends on the platform and OS version. Starting from Android 8, the background execution of applications is limited to several minutes after the last user interaction with the application [Bac18]. In Android 9, sensors cannot be accessed in the background by default [Anda]. To be able to continuously record sensors in the background on Android 8 and 9, the attacker needs to declare a so-called *ForegroundService* [Sen18], which results in a visible user notification. This notification, however, can be masqueraded as a seemingly benign functionality which needs to be constantly running, e.g., a fitness activity tracker.

Similarly to Android 8 and newer, execution of iOS applications is suspended shortly after being moved to the background (starting from iOS 5). Furthermore, the iOS platform

does not provide a functionality to keep the background application active for an arbitrary amount of time. However, execution time can be granted to background applications when they perform specific set of actions, e.g., playing audio, receiving location updates, processing updates from a server or reacting to remote push notifications [App20].

Nevertheless, in all cases, the attacker would be able to classify applications or webpages opened shortly after the victim leaves a malicious application, classify background activities, or activity opened in split screen mode. Table 6 summarizes the differences between the in-app and in-browser scenarios.

4.3.2 *Additional assumptions*

Following other works on website and application fingerprinting, we discuss several additional assumptions [Juá+14]. In this section, we reason about these assumptions with regard to our scenarios, and show that many of them can be encountered on modern mobile platforms, in comparison to traditional desktop systems.

First, it is typically assumed that users open applications (websites) sequentially and have only a single *active* application (website) open at a time. This assumption is reasonable for our scenarios, as on mobile devices a user can not keep more than two applications in the foreground at a time, with two only in split-screen mode. Furthermore, modern mobile browsers significantly limit JavaScript execution in background tabs or even completely prevent it, to reduce power consumption. As a result, in a general case only one application remains active at a time, and, in case of a web browser, only one tab can be active.

Second, we assume that there is no user-invoked activity in the background. As described in Section 4.3.1, modern Android and iOS systems limit execution time of background processes. We confirmed that these limitations result in low average background activity. We performed the test measurement of the average CPU activity over a period of 24 hours on two devices with numerous applications installed and the recording application in the foreground. As a result, we obtained the average CPU utilization of only 1.9%, with the standard deviation of 1.7%. Furthermore, in the course of our experiments we did not take any measures to specifically prevent background activity. We performed our measurements on unmodified smartphones, with up to 60 additional popular applications installed. These applications could potentially generate CPU noise in the background during the continuous recording (over 30 hours of recording per device and tested scenario). Nevertheless, the high classification rates show that these activities do not significantly affect the recording traces. Overall, we can expect that the impact of background activity on the classification is low.

Third, we present evaluation results under the assumption that websites (applications) do not change over time. As observed in our experiments and other works (e.g., see [Juá+14; Yan+17]), this assumption does not hold for websites, and the attacker needs to periodically re-run the learning phase. However, we observed that traces from applications remain stable unless they get updated. In addition, configuration options of the browser are comparably limited on mobile devices, so it is easier for the attacker to replicate the user client-side settings.

Finally, it is generally assumed that the attacker can detect the beginning and end of each activity to be classified. In practice, this can be hard to achieve: In our case, *any*

CPU activity performed on a device can cause magnetic disturbances. As one potential solution, we show in Section 4.4.4 that the attacker can identify potential time points when the target activity could have started by computing the cross-correlation with the predefined pattern, and run the classification only at these specific points.

Apart from these assumptions, typically addressed in works on website and application fingerprinting, in this work we additionally assume that the victim is not actively moving the device, as movements affect magnetometer data. In Section 4.5.5, we evaluate the impact of minor movements on the classification accuracy when the smartphone is being held in hand. Furthermore, in Section 4.4.5, we propose an approach how the attacker can identify and filter out sensor readings which are disturbed by movements.

As a result, we believe that our scenarios are realistic under given assumptions.

4.4 METHODOLOGY

In this section, we discuss implementation details about how data was collected, its pre-processing, feature extraction and classification, describe approaches to identify the target activity in the continuous measurement stream and to identify traces disturbed by device movements.

4.4.1 *Data collection*

To collect a large set of labeled traces in the learning phase, we trigger opening applications and websites from our datasets in an automated and controllable way, using the Android Debug Bridge (*adb*) [Andb] tool from the Android SDK. Our service script opens each application from the dataset, waits for a predefined duration, and closes the target application. Similarly, for website fingerprinting, the service script opens the Chrome browser and the corresponding website. Additionally, we implement opening websites in a separate application with an embedded WebView component. It allows us to evaluate the website fingerprinting in the cloud testing platforms, such as the AWS Device Farm [AWS20]. These platforms allow developers to test mobile applications remotely on multiple devices. However, they do not provide access to devices through the *adb*. Therefore, we could not evaluate the application fingerprinting or use the Chrome browser on these platforms.

To collect resulting magnetometer disturbance traces, we implemented an Android application which runs in the background, records 3-axis magnetometer data, and sends it to the attacker-controlled server. Similarly, for the in-browser scenario, we implemented a web page which records the sensors using the Generic Sensor API in the mobile Chrome browser, and sends the data to the server. As a result, for each opened application or website, the server receives labeled (in the learning phase) or unlabeled (in the testing phase) sensor measurements.

4.4.2 *Data preprocessing*

Subsequently, we convert the raw 3-axis data trace into a discrete-time one-dimensional trace. For this purpose, we apply Principal Component Analysis [Hot33] to the data, choosing the first component as the result. The resulting data represents the one-

dimensional axis with the highest data variance. Assuming that the orientation of the device is not changed significantly and that the ambient magnetic field together with EM noise is constant at a given point in time, this variance represents the vector of the EM emanation caused by the CPU. The disturbance in the one-dimensional trace can be directed above or below the baseline level. Therefore, we add both the original recorded trace and its inverse with regard to the baseline to the dataset in the training phase of the classifier, considering both traces as representations of the corresponding CPU pattern.

Finally, we normalize the result to the range [0–1], so the resulting values do not depend on the maximum possible amplitude of the disturbance (which is device-specific, see Section 4.5.1). Instead, the result contains information about the "shape" of the pattern, which represents the unique CPU activity pattern.

4.4.3 *Feature extraction and data classification*

Finally, we divide the resulting normalized discrete-time values of the axis with the biggest variance into equal-size overlapping intervals (bins) and calculate the mean value within each bin. These mean values are used as features for classification. To classify the traces, we use a Random Forest [Bre01] machine learning classifier, as it outperforms other algorithms in our experiments in terms of resulting classification accuracy. We split the dataset into training set (80%) and test set (20%). The 5-fold cross-validation is performed on the training set to select optimal hyperparameters using the grid search, which include the number of estimators in the forest, the maximum number of features, maximum depth of the tree, and minimum impurity decrease. The test set was only used to compute the accuracies when evaluating the classifier in our experiments.

In our experiments, we use the RandomForestClassifier from the *scikit-learn* library [Ped+11] to perform classification. The values of the hyperparameters selected after the cross-validation are: *n_estimators* = 1100, *max_features* = *log2*, *max_depth* = 50, *min_impurity_decrease* = 0.0001. Other hyperparameters are kept as default.

4.4.4 *Identifying target activity during continuous usage*

As we discussed in Section 4.3.2, the attacker is assumed to know the beginning of the activity to be classified. In our case, the attacker needs to continuously monitor magnetometer disturbances, which can be caused by *any* application.

However, if the practical goal of the fingerprinting is to identify whether the victim opens a particular target application or a website (or set of or websites), we propose the following approach to reduce the amount of data to be processed by the classifier. First, the attacker can compute an averaged CPU activity pattern for the target application or website by computing mean values along multiple traces for this activity (known from the learning phase). Then, this pattern can be used to calculate the cross-correlation with the continuously recorded data using the following formula:

$$c_{tp}[k] = \sum_n t[n+k]p[n],$$

where t is a recorded discrete trace and p is the computed pattern. If the target activity was produced within the recorded interval, a strong peak is present in the cross-correlation

result at the corresponding time point. In practice, however, due to noise and slight changes in the produced activity patterns, cross-correlation results will not have a single strong peak, but multiple potential peaks. However, due to similarity in actual and averaged patterns, one can expect that the actual time point corresponds to one of these peaks. Therefore, the classification can be run only at time points where peaks are present in the cross-correlation result with a predefined threshold. This threshold sets a trade-off between the number of peaks and the accuracy of peak detection. We evaluate this approach in Section 4.5.4.

Interestingly, for website fingerprinting, an attacker can also perform this step on recorded data to first detect the web browser application to be opened (as an application fingerprinting task), and then classify the recorded interval after the browser was opened.

4.4.5 *Identifying device movements*

If the victim rotates the device, a corresponding change in the global orientation and relative direction to the magnetic north will cause a shift in magnetometer readings along three axes. In this case, the PCA-based trace will no longer represent disturbance exclusively caused by CPU activity. To identify and filter out traces which are affected by movements, we propose to analyze the rotation rate measurements from the gyroscope sensor simultaneously with the magnetometer. Access to gyroscope also does not require permissions, and its data is not affected by the CPU activity. Therefore, the attacker can use gyroscope readings to estimate if the device has been significantly moved.

More specifically, we propose two criteria for identifying traces affected by movements. The first criterion is the mean amplitude of the rotation rate along all three axes, which indicates the overall presence of movements within the recorded interval. The second criterion is the highest amplitude of the rotation rate, which indicates abrupt change in orientation. If the value computed for any of two criteria exceeds a predefined threshold, the recorded trace is considered to be affected by movements, and the trace can be ignored during the classification. In Section 4.5.5, we evaluate this approach for the smartphone being held in hand.

4.5 EVALUATION

In this section, we identify devices on which magnetometers are affected by the CPU, evaluate the classification performance, show the success rate of capturing the target activity, and investigate the impact of minor movements.

4.5.1 *Information leakage*

In this experiment, we examined whether the magnetometer readings on mobile devices are affected by the CPU workload. For this purpose, we produced a predefined CPU activity pattern on a device and analyzed resulting sensor disturbances. The pattern consists of alternating high and low CPU loads lasting for 2 seconds. To produce high loads, we concurrently ran so-called busy waiting loops in a number of threads, equal to the number of available logical cores on a device, utilizing up to 100% of the CPU time. To produce low loads, we paused the execution.

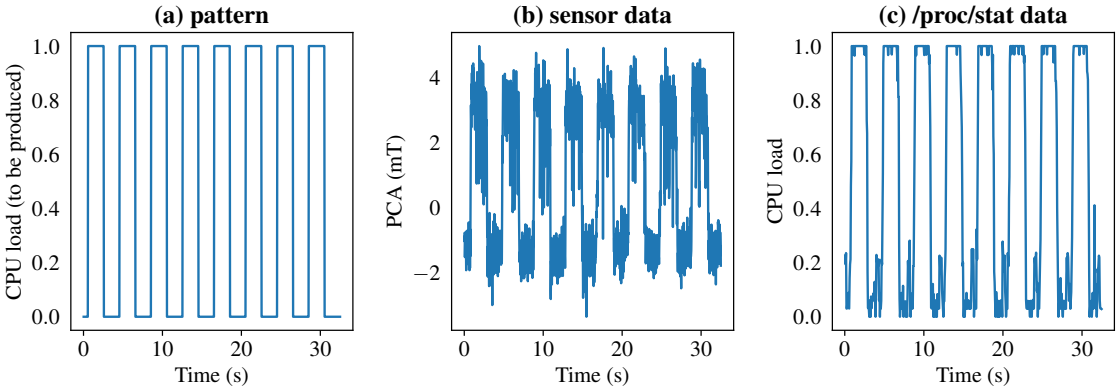


Figure 15: Example of the expected CPU pattern to be produced **(a)**, recorded sensor data **(b)**, and actual CPU pattern recorded through `/proc/stat` **(c)**.

Afterwards, we calculated the correlation between this pattern and recorded measurements. If the device runs Android 7 or iOS, we were able to additionally calculate the correlation coefficient with the actual produced CPU activity pattern, recorded using */proc/stat* or *host_processor_info*, respectively. Some examples of predefined pattern and corresponding magnetometer and */proc/stat* recordings are illustrated in Figure 15. We also measured the signal-to-noise ratio (SNR), i.e., the ratio between the average amplitude of the disturbance caused by the high CPU load and the standard deviation of measurements without CPU activity. It allows us to estimate how robust the produced disturbance is against environmental and intrinsic noise. To evaluate a large number of devices, we conducted measurements using two cloud platforms, Visual Studio App Center [Vis20] and AWS Device Farm [AWS20]. We selected all available devices running Android 7 or higher, and all devices running iOS 11 or higher. Additionally, five devices were used in the lab in a typical office environment. We could not control the environment of the devices in the cloud (such as noise), and tested them as is.

We found that magnetometers on 56 out of 80 devices are affected by the CPU activity. Results for selected devices are shown in Table 7, the full list is provided in Appendix A.2. On these devices, the disturbance clearly correlates with the CPU activity (with correlation scores over 80% on average). On most of the devices, the signal exceeds noise. In further experiments, we confirmed that an SNR of ≈4dB is sufficient to establish fingerprinting. Magnetometers on other 24 devices, listed in Appendix A.2, however, were not affected by CPU activity.

As one can see from both tables, the sensor model does not indicate whether the magnetometer is affected by the CPU: For example, sensors AKM AK09915 and AKM AK0991X can be found on both affected and not affected devices. We believe that the reaction mostly depends on the physical location of the sensor with regard to the CPU and power wires, and applied shielding.

As a result, we believe that the attack is practical, since modern popular devices (e.g., recently released smartphones Google Pixel 3, Samsung Galaxy S10, and iPhone XS) are all affected.

Table 7: Selection of devices on which sensor measurements correlate with CPU activity. The full list of affected devices is presented in Appendix A.2. The table shows the cross-correlation between sensor data and expected CPU activity pattern (Corr.Pattern); for Android⩽7 and iOS, also between sensor data and *actual* CPU loads (Corr.CPU), as well as the SNR.

Smartphone	Setup[a]	Magnetometer[c]	Correlation		SNR,
			Pattern	CPU[b]	dB
Android					
Google Pixel	V,A,L	AKM AK09915	0.86	0.89	14.7
Google Pixel 2	V,A,L	AKM AK09915	0.78	—	10.8
Google Pixel 3	V,A	STMicro LIS2MDL	0.90	—	14.2
Google Pixel C	V	Google CROSEC	0.91	—	27.4
Google Pixel XL	V,A	AKM AK09915	0.83	0.95	12.2
Huawei Mate 20 Pro	V	AKM	0.81	—	20.1
HTC U Ultra	V	AKM AK09915	0.95	0.96	28.6
HTC U12+	V	AKM AK09915	0.60	—	7.1
LG Nexus 5X	V,L	Bosch BMM150	0.88	0.93	15.5
LG V30	V	AKM LGE	0.93	0.96	22.6
OnePlus 3	V	MEMSIC MMC3416PJ	0.92	0.95	14.7
Samsung Galaxy Note 9	V,A	AKM AK09918C	0.52	—	4.1
Samsung Galaxy S9+	V,A	AKM AK09916C	0.55	—	4.2
Samsung Galaxy S10	V,A	AKM AK09918C	0.65	—	7.9
Sony Xperia 10 Plus	V	GlobalMEMS GMC306	0.78	—	8.8
Xiaomi Mi A1	V	AKM AK09918	0.82	—	11.0
iOS					
iPad Air 2	V,A	Unknown	0.84	0.42	13.0
iPad Mini 3	V	Unknown	0.95	0.96	16.8
iPad Pro 12.9	V,A	AKM AK8789	0.93	0.63	16.3
iPhone 5S	V,A	AKM AK8963	0.89	0.80	12.1
iPhone SE	V	Alps HSCDTD007	0.91	0.87	19.2
iPhone 6	A	AKM AK8963	0.70	0.59	8.4
iPhone 6S	V,A,L	Alps HSCDTD007	0.81	0.81	20.3
iPhone 7	V,A,L	Alps HSCDTD008A	0.89	0.85	11.0
iPhone 8 Plus	V,A	Alps e-Compass	0.87	0.81	12.0
iPhone X	V,A	Unknown	0.77	0.74	22.5
iPhone XR	V,A	Unknown	0.88	0.86	16.9
iPhone XS	V,A	Unknown	0.75	0.72	12.1

[a] V — Visual Studio App Center; A — AWS Device Farm; L — lab

[b] CPU utilization data is available only on devices running Android ⩽ 7 (over */proc/stat*) and iOS (over *host_processor_info*).

[c] For Android devices, information is available from the *Sensor* API. For iOS devices, information from publicly available online resources is used.

Table 8: Classification accuracy for website and application fingerprinting in the in-app and in-browser scenarios compared to classification using `/proc/stat` data. Traces have been collected on a Google Pixel 2 smartphone in the lab environment.

Dataset	Setup	Browser	Sampling Rate, Hz	Accuracy, %
Website fingerprinting				
sensor	in-app	Webview	100	90.5
sensor	in-app	Chrome	100	74.9
sensor	in-browser	WebView	10	86.7
cpu	in-app	Webview	50	89.0
Application fingerprinting				
sensor	in-app		100	90.0
cpu	in-app		50	95.8

4.5.2 *Classification results*

In this experiment, we evaluated the classification accuracy of our attack in a so-called *closed-world* scenario, when the attacker aims to identify the visited website (application) among a predefined list of websites (applications).

For website fingerprinting, we collected magnetometer and CPU utilization traces during retrieval of the 50 most popular websites from the Alexa Top 500 Global Sites list [Ale18], merging websites with multiple domains together (e.g., google.*). We collected 175 traces per website, with a duration of 12s each. Similarly, for application fingerprinting, we collected traces of 65 applications being launched, 175 traces per application, with a duration of 12s each. The applications were taken from the list of popular Android applications [And18a]. A full list of used websites and applications, as well as classification results for individual websites and applications, are provided in Appendix A.2. All traces were collected on a Google Pixel 2 smartphone lying on a table in the office environment.

Afterwards, we ran the classification using both sensor and */proc/stat* data. The results in terms of classification accuracy are shown in Table 8. As we can see, the classifier performs with an accuracy of over 80% for website and application fingerprinting. Notably, the proposed approach has a similar performance in comparison to the classification based on actual CPU activity collected through */proc/stat*. These results indicate that the magnetometer-based side channel leaks sufficient information about CPU activity.

Classification accuracies for different setups are also compared in Table 8. More specifically, we separately tested website retrieval in an embedded WebView component with cache disabled, as well as using a full mobile Chrome web browser with cache enabled. As one can see, the classification results are similar for both cases. However, the caching does affect the resulting patterns. We also achieved 86.7% accuracy with web-based recording of sensors using Generic Sensor API, which proves the applicability of our method to the in-browser scenario.

Additionally, we evaluated the classifier on a larger dataset. We increased the number of websites to 100 and repeated the experiment on a single device in the in-app scenario

Table 9: Classification accuracy for website fingerprinting in the in-app scenario for several smartphones, for intra-device and inter-device modes.

Device	Setup[a]	Accuracy, %	
		intra-device	inter-device
Google Pixel XL	V	62.5	53.2
Google Pixel 2	L	90.5	83.4
Google Pixel 3	V	83.6	80.8
HTC U12+	V	86.6	80.9
Samsung Galaxy Note 9	V	86.4	82.0
Samsung Galaxy S9+	V	81.9	78.1

[a] V — Visual Studio App Center; L — lab

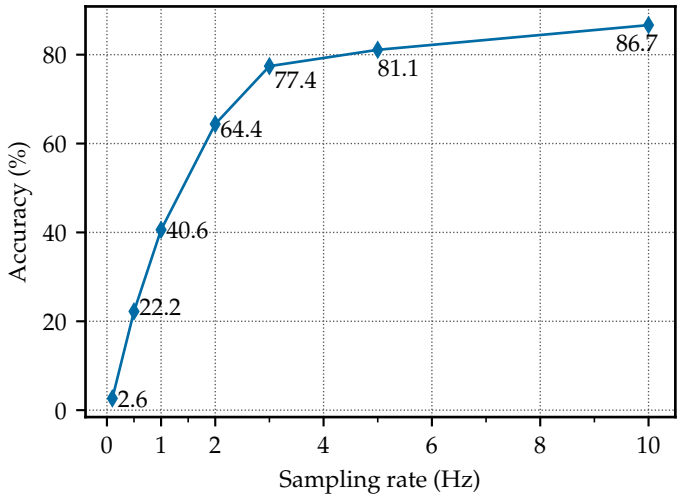


Figure 16: Classification accuracy of website fingerprinting depending on a sampling rate.

with an embedded WebView component. The classifier performed with an accuracy of 87.6%, comparable to 90.5% in the initial setup (see Table 8).

Afterwards, we ran the website fingerprinting experiment on five other smartphones in the cloud environment. We calculated the success rates for intra-device (with a training and testing performed on individual devices) and inter-device (with a training phase performed on traces from all devices, and testing on individual devices) modes. The results are summarized in Table 9. Google Pixel XL and Samsung smartphones performed worse than other devices due to the lower sampling rate and the lower SNR, (see Section 4.5.1), respectively. The activity patterns are also device-specific. Therefore, the attacker may need to train the classifier on numerous devices or take into account the target device model.

Finally, we evaluated how the sampling rate of sensor data gradually affects the classification accuracy. For this purpose, we further decreased the sampling rate for the dataset of websites recorded in the in-browser scenario and calculated the resulting classification accuracies. Figure 16 shows the results. As one can see, the sampling rate needs to be reduced to less than 1 Hz in order to make the attack impractical.

Table 10: Classification results for the open-world scenario, in terms of the precision and the recall for the five monitored websites.

Website	Precision,%	Recall,%	F1 score,%
facebook.com	95.5	48.8	64.6
google.com	100.0	41.9	59.0
taobao.com	84.6	51.2	63.8
wikipedia.org	77.1	86.0	81.3
youtube.com	97.3	83.7	90.0
Average (monitored)	90.9	62.3	71.7
Average (overall)	92.2	68.6	76.3

4.5.3 *Open-world scenario*

In this section, we evaluate our classifier in a so-called *open-world* scenario. In comparison to the *closed-world* scenario, (evaluated in Section 4.5.2), a victim can visit a much larger set of websites not known to the attacker. Consequently, the attacker cannot generalize the classifier and identify every visited website. Instead, the attacker aims to identify whether a victim visits specific websites, further referred to as *monitored* websites. To perform the experiment, we first collected traces for 50 most popular websites, 175 traces per website, to train the classifier similarly to the closed-world scenario. However, in this case, we selected five the most popular websites to be five monitored classes. Other 45 websites were labeled as *not monitored*, i.e., they belonged to a separate class. For testing, we used another list of popular websites [Maj18], which is larger than the Alexa list. We collected one trace for each of the 7,500 most popular websites, excluding 50 websites (or their alternative domains) used in the training phase. Finally, we collected 40 traces of each of the five monitored websites, to have a total of 7,700 traces in the testing set. All traces were collected on a Google Pixel 2 smartphone in the in-app recording mode.

To evaluate the results, for each class we calculated the *precision* and *recall* using the following formulae:

$$\mathrm{precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}, \quad \mathrm{recall} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}},$$

where TP (True Positives) is the number of correctly classified traces of the considered class, FP (False Positives) — the number of traces of other classes which are incorrectly classified as the considered class, FN (False Negatives) — the number of traces incorrectly classified as the other class. The previously used overall classification accuracy (99.6% in this case) is not a practical metric for the open-world experiment, as the classes are imbalanced, i.e., the number of traces for non-monitored websites significantly exceeds the number of traces for monitored websites.

Table 10 shows the classification results. We can see that the average achieved recall of 68.6% is lower in comparison to the closed-world scenario, but is still practical. The precision is, however, relatively high: 92.2% for all websites and 90.9% for monitored classes. The high precision is especially valuable in the open-world scenario, as it ensures the attacker that the victim did visit the monitored website if it was identified by the

classifier. As a result, we believe that our approach is applicable to the open-world scenario.

4.5.4 *Continuous usage*

In this experiment, we evaluated the ability of the attacker to detect the starting point of the trace to be classified in a continuous recording stream. The detection is performed by calculating the cross-correlation with the predefined pattern, as we described in Section 4.4.4. We evaluated the approach in the scope of application fingerprinting and chose the Chrome browser as the target activity. We made 50 continuous recordings lasting 100s each, and within every recording we opened the target application and two other applications at specific non-overlapping time points. The applications for each recording were randomly chosen from the dataset. This way, traces contained the pattern corresponding to the target application, as well as noise from other activities. For each recording, we calculated the cross-correlation between the recorded trace and a pattern computed for the target application.

Then, we detected local maxima (peaks) in the result. The set of peaks was filtered according to three threshold parameters: peak height, prominence and width. We considered a peak as true positive if it was discovered within a 1s-interval around the time point when the target application was actually opened. Other detected peaks were considered as false positives. A false negative was assumed if there was no peak within the corresponding interval. In the end, we calculated the classification precision and recall according to these definitions, to indicate how effectively cross-classification can narrow the search area. For 50 recordings and our set of parameters, we achieved a precision of 24.5% and 72.9% recall. The attacker can vary parameters of the cross-correlation to increase the recall at the expense of precision (i.e., discover more peaks, including false positives), and vice versa.

Finally, we ran the classification at all discovered time points including false positives. This step ensures that the cross-correlation approach in the first step identifies the time points with sufficient precision, so that the classifier can correctly identify the true positive samples. The accuracy in our experiment reached 81%, which is comparable to the 90% achieved in the closed-world experiment with a known beginning point. The decrease is observed due to a number of false positives at time points corresponding to noise, as the classifier in the closed-world scenario has not being trained on noise data. As a result, the experiment shows that the attacker can efficiently reduce the amount of data to be processed using the proposed approach.

4.5.5 *Robustness to movements*

In this experiment, we evaluated the classification accuracy when the smartphone is being held in hand, and our approach to identify traces affected by movements described in Section 4.4.5. We used the classifier trained for the website fingerprinting in the closed-world scenario on a static device (see Section 4.5.2). Afterwards, we recorded a total of 500 test traces while freely holding a smartphone in hand.

When the classifier was applied to the whole test dataset without filtering, the overall accuracy dropped to 64.8%, indicating that movements do affect the measurements.

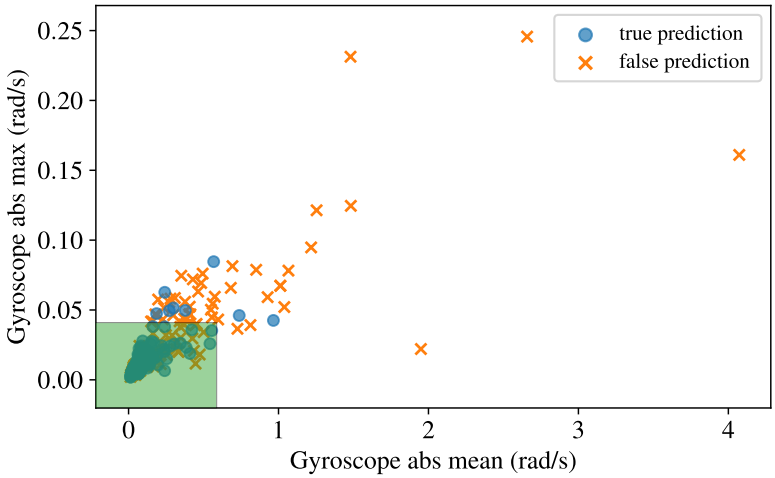


Figure 17: Distribution of traces with regard to their gyroscope-based metrics for movements. Numerous wrongly classified traces lie outside the highlighted threshold area.

However, wrongly identified traces could be filtered out using the proposed approach: in Figure 17, one can see that for numerous wrongly classified traces the thresholds for indicating movements are exceeded. By applying the filtering based on the thresholds before the classification, 21% of the measurements were identified as affected by movements. After removing affected traces from the dataset, the accuracy reached 73.3%. The accuracy is lower in comparison to the accuracy achieved for the static device (90.5%), but remains practical. Nevertheless, a larger user study and more detailed analysis of the impact of movements may be needed to prove the wide applicability of the approach.

4.6 COUNTERMEASURES AND DISCUSSION

In this section, we discuss as some related aspects and directions for future work.

COUNTERMEASURES. There are several possibilities to prevent the presented information leakage through magnetometer disturbance. First, as mentioned in Chapter 2, physical shielding of the CPU would be the most straightforward way to limit the susceptibility of the sensor to electromagnetic interference. However, this measure opposes an industry trend of making smartphones thinner and lighter, and cannot protect existing devices from the attack. Furthermore, as we have discovered in our experiments in Section 4.5.1, some smartphones and tablets actually do not react to CPU activity, presumably due to the sensor location relative to the CPU or power supply components. We, therefore, believe that the location of the sensors should be taken into account when designing the layout of the smartphone motherboard. Second, based on our evaluation in Section 4.5.2, further limiting the sensor sampling rate to 1 Hz significantly reduces the classification accuracy of fingerprinting. However, with such a lower sampling rate it may be still possible to infer information about more coarse-grained activities. Furthermore, it may negatively affect the performance of legitimate applications. Third, an explicit user permission can be introduced to limit access to magnetometers. However, users may not correctly perceive potential privacy threats emerging from sensors in mobile

devices [Meh+18]. Therefore, an explanation of potential risks might be needed. Moreover, a lot of mobile devices in use run outdated operating system versions [Dis18]. Finally, to limit the attack surface of our attack, access to magnetometers can be restricted for applications opened in the split-screen mode and can immediately be blocked when the application goes to the background.

The described countermeasures would require hardware or software changes, may have performance or production cost drawbacks, and require careful design decisions. In particular, we are concerned about the ongoing deployment of the Generic Sensor API in browsers as access to the magnetometer from the browser significantly extends the attack surface. We recommend requiring an explicit permission to access the magnetometer on web pages. An alternative recommendation would be to further reduce the sampling rate.

AGING OF DATA. As shown in other works on website fingerprinting (e.g., [Juá+14; WG13]), aging of sampling data affects classification accuracy. Therefore, the attacker needs to repeat the learning phase periodically. One interesting direction for future work would be a detailed investigation of which elements on a web page affect the classification the most when being changed, for different fingerprinting methods. For example, increasing the web page size by extending the text content can affect fingerprinting based on traffic analysis, but may have no substantial effect on the sensor disturbance in our approach, since text rendering is computationally inexpensive for the CPU.

INFLUENCE OF EXTERNAL NOISE. In principle, magnetometer sensors are susceptible to external electromagnetic noise. However, as we have shown in Chapter 2, magnetometers are affected by the noise from nearby computers only at short distances ($\leqslant$15cm). We performed all experiments in a typical office environment with natural arrangement of multiple electronic devices, such as laptops, Wi-Fi access points, and other smartphones. As our results indicate, activity of these devices did not impair the performance of our approach. Nevertheless, systematic analysis of potential external noise sources and their impact on classification can be performed as future work.

COMBINING SOURCES OF LEAKAGE. Finally, it would be interesting to combine our approach with works exploiting other side-channel information on smartphones, especially leakages of other nature such as memory access statistics. In this way, a feature set combining different side-channel information can potentially improve the classification accuracy.

4.7 SUMMARY

In this chapter, we presented a method to utilize susceptibility of magnetometers to CPU activity of mobile devices to identify running applications and websites on mobile devices. We confirmed that cross-component interference between CPU and magnetometer is observed on a large number of modern smartphones, and demonstrated that this information leakage is sufficient to identify opened websites and applications. The presented method does not require any user permissions, can be run in both in-app and

in-browser scenarios, and therefore further demonstrates privacy risks emerging from sensor-based side- and covert channels.

5

SENSOR-BASED ACOUSTIC COVERT CHANNEL

In this chapter, we introduce a covert channel that utilizes the reaction of gyroscope sensors in smartphones to ultrasonic acoustic interference. As a result, the sensor is capable of capturing acoustic signals inaudible to human ears at a distance. Unlike microphones, access to gyroscopes up till now does not require explicit user permissions. Therefore, the exploited reaction allows attackers to establish *zero-permission* acoustic covert channels. Furthermore, we introduce the application scenario of cross-device tracking, demonstrating how covert channels can be used to link activities on two devices to the same user, attacker user's privacy.

Remarks. Content in this chapter is based on the corresponding publication [MSK18].

5.1 MOTIVATION AND CONTRIBUTIONS

Commercial companies today collect an increasing amount of information about their users, to improve customer experience, but also to increase financial profits by showing targeted advertisements. Advertising components can be embedded as third-party content on hundreds of websites or TV streams, making it possible to analyze user activity on a large scale. Moreover, the widespread use of mobile and wearable devices has resulted in the demand for *cross-device* tracking technologies, which allow companies to correlate user activities even across different devices. This introduces a serious privacy threat, since such aggregated user profiles may contain sensitive information about personal interests, location, health, beliefs or sexuality, while users remain unaware of the scope and mechanisms of such tracking [RKM17; Cal+15a].

Usually cross-device tracking is performed by linking the device to some deterministic information provided by users themselves, e.g., application or website login credentials [Bro+17], or by comparing attributes shared by all the devices, such as IP addresses or location data. Recently, ultrasonic cross-device tracking (uXDT) has emerged, based on embedding tracking identifiers into ultrasonic sounds and detecting them with a microphone on a user's smartphone, essentially creating an inter-device acoustic covert channel. In particular, companies like Shopkick, Lisnr and Signal360 provide a way to deploy ultrasonic beacons at specific locations (e.g., shops or festivals) and detect them in a mobile application to show location-relevant content. The company Silverpush developed means of embedding tracking identifiers into TV streams. Meanwhile, researchers [Mav+17] demonstrated how the uXDT technology can be used for web-tracking purposes, e.g., for transmitting a tracking ID from the Tor browser to an application on the user's smartphone in order to de-anonymize the web session.

Due to privacy concerns, uXDT technology raised the attention of public media [Bre18] and the security community [Arp+17; Mav+17]. In response, the Federal Trade Commission issued warning letters to app developers who use Silverpush components, asking them to explicitly disclose the usage of ultrasonic tracking. Nevertheless, researchers recently found 234 Android applications that are listening in the background for ultrasonic

beacons from TV streams [Arp+17], some of them with millions of users, proving that the technology is being actively deployed.

Starting from Android 6, mobile applications are required to ask for a user permission at run-time to access the microphone (in the past it was only done during app installation). This way, Android devices prevent stealthy audio recording. On iOS devices, a pop-up warning is additionally shown when the app accesses the microphone in the background. This way, an attempt to start detection of uXDT signals by the app will most likely raise user's attention.

In this chapter, we present a new approach to perform uXDT, which does not require access to a microphone at all, and instead uses gyroscopes in smartphones or smartwatches as receivers for ultrasonic signals. It has been shown that microelectromechanical (MEMS) gyroscopes are susceptible to acoustic vibrations at specific *resonance* frequencies [Son+15; Far+16], typically within ultrasonic range (19–29kHz). In our work, we show that ultrasonic signals can be emitted at these frequencies with commonly-used audio hardware, and subsequently be captured at a distance by gyroscopes of modern smartphones and smartwatches. By analyzing spectral characteristics of gyroscope's response to sound, the signal can be decoded even in the presence of device movements, e.g., when a smartphone is held in a hand, or a smartwatch is worn on a wrist.

We show that cross-device tracking can be established between commonly-used devices (e.g., a laptop and a smartphone) at distances of up to 35cm using internal laptop speakers at 75% volume level, achieving a bitrate of 10bit/s. With a more powerful speaker, distances of several meters and a bitrate of up to 20bit/s are achieved. Although distance and bandwidth are limited in comparison to existing uXDT solutions with recording using microphones, the proposed method can be run completely stealthily to users, as access to gyroscope data does not require any explicit permissions. Adding such a permission, as well as applying other possible countermeasures (as discussed in Section 5.6), introduces technical and usability problems and may not completely eliminate this new attack vector. Therefore, our work demonstrates that even with the hardened permission model on mobile devices, uXDT technologies still pose a significant privacy risk.

CONTRIBUTIONS. Our contributions can be summarized as follows:

- We introduce a novel method of cross-device tracking through an inter-device covert channel that exploits the sensitivity of gyroscopes in mobile devices to acoustic resonance. Unlike existing approaches using microphones as receivers of acoustic covert-channels, our method runs unnoticeably to users without any explicit permissions, posing significant privacy risks.

- We present an implementation of the covert channel with an encoding scheme that is robust to background noise and natural device movements, and demonstrate its applicability to cross-device tracking scenarios, including web tracking, TV media tracking and location tracking.

- We evaluate the implementation on different hardware setups. To the best of our knowledge, our work is the first to consider and evaluate acoustic transmission using MEMS gyroscopes as receivers *at a distance*, when sensor is not directly adjacent to the source.

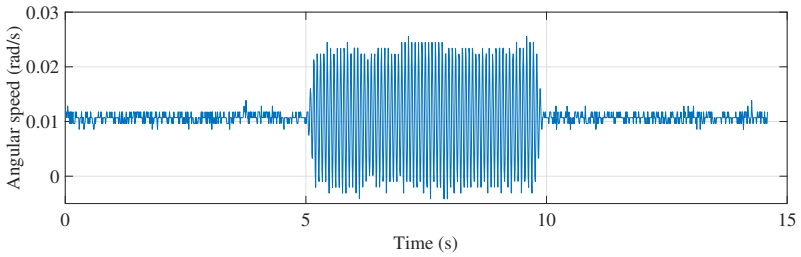


Figure 18: Example of acoustic disturbance of gyroscope measurements over the x-axis, due to playing a 27kHz sine wave (5–10s). Measurements are recorded on an iPhone 6s in a stationary position, located 30cm from the speaker.

5.2 BACKGROUND

In this section, we briefly describe use of MEMS gyroscopes in mobile devices, their susceptibility to acoustic vibrations, and abilities of modern audio hardware to transmit covert acoustic signals.

5.2.1 *Susceptibility of gyroscopes to ultrasonic sounds*

Most modern smartphones and smartwatches are equipped with gyroscopes, which measure the rotation rate of the device in radians per second around three physical axes (often referred as the pitch, roll and yaw), to estimate its orientation in space. The acquired data is used in games, virtual reality applications, fitness trackers, etc. In Android and iOS devices, 3-axis gyroscope values are retrieved by using the Sensor API [Sen18] and the Core Motion [App18b] framework, respectively. The sampling rate of the measurements depends on the sensor, and is additionally limited by the operating system to reduce power consumption. On all the tested devices, we were able to access gyroscope data with sampling rates of 60–500 Hz.

The underlying principle of the MEMS design is based on Coriolis force acting on a moving sensing mass. Usually gyroscopes consist of one or several sensing masses, constantly vibrating at a specific *resonance frequency*. When the gyroscope is rotated, the Coriolis force is applied to the sensing mass orthogonally to its vibration direction and the rotation axis, with an amplitude proportional to the rotation rate. A detailed explanation of MEMS gyroscope design can be found in [Kaa09].

More importantly for our work, it is known that MEMS-based gyroscopes are susceptible to acoustic signals at frequencies close to the sensor's resonance frequency [Cas+07; Dea+07]. The acoustic waves cause the sensing mass to additionally vibrate on the axes corresponding to the Coriolis force direction, and disturb the resulting measurements on one or all of the axes. For example, Figure 18 demonstrates gyroscope measurements from a stationary iPhone 6s placed near a speaker. Once 80dB sound is played at gyroscope's resonance frequency starting at 5s, a clear disturbance in gyroscope measurements is noticeable.

To limit acoustic disturbance from background noise, manufacturers design the sensors to have a resonance frequency in ultrasonic range. Most of the gyroscopes tested in other works [Son+15; Far+16] and in our experiments (Section 5.5.1), had a resonance frequency within 19–29 kHz. This fact makes gyroscopes suitable receivers for covert

ultrasonic signals: humans can perceive sounds only within the 20 Hz–20 kHz range, with the upper threshold declining with age, so most people over 18 years cannot hear frequencies above 16 kHz [HSM94].

5.2.2 *Capabilities of commodity audio hardware*

In order to perform uXDT by disturbing gyroscopes at resonance frequencies, the transmitter should be able to produce ultrasonic signals at 19–29 kHz. According to the Nyquist sampling theorem, the highest possible frequency of the digital signal to be reproduced without aliasing should not be more than half of the sampling rate. Sound interfaces of computers and mobile devices typically support at least 44.1 kHz, and most of them even a 48 kHz sampling rate, to comply with popular audio codecs. Moreover, many digital-to-analog converters (DAC) and speakers in modern computers, home theater systems and smartphones have 96 kHz or even 192 kHz sampling rates to support *high-resolution* audio [Hir20; Jap20]. Therefore, ultrasonic signals of up to 24 kHz can natively be generated by most commodity audio hardware, and many consumer devices are able to reproduce sounds of higher frequencies, covering the aforementioned 19–29 kHz range. Additionally, the abilities of audio hardware to produce ultrasonic sounds are limited by its non-linear frequency response: typically speakers attenuate the signal at frequencies higher than 20–22 kHz, as they are not perceived by humans. Nevertheless, in our experiments we show that produced sound pressure level is sufficient to cause acoustic disturbance of the gyroscope at a distance.

5.2.3 *Related work*

ACOUSTIC CROSS-DEVICE TRACKING. The idea of ultrasonic communication has been explored in research over the last years, mainly focusing on establishing covert channels between isolated computers [HG13; HG14; CA14] and mobile devices [Des14] by using their speakers and microphones. When commercial solutions of tracking TV ads and user location using ultrasonic beacons emerged on the market and raised public attention, researchers started to investigate the security and privacy implications of such technology. Mavrodius et al. [Mav+17] described several potential uXDT-based attacks, and designed a browser extension to filter out high frequencies from audio playback, as well as an Android permission to provide fine-grained control over microphone recordings. Arp et al. [AQW16; Arp+17] presented a detailed analysis of Silverpush and Lisnr implementations, and found 234 existing Silverpush Android applications that are listening in the background for ultrasonic beacons from TV streams, proving that the technology is being actively deployed in the wild. In our work, we demonstrate that described attacks may pose even more significant privacy risk, as they can be established fully unnoticeably to users, utilizing zero-permission access to gyroscope sensors on mobile devices.

MEMS SENSORS' REACTION TO ACOUSTIC VIBRATIONS. Table 11 summarizes prior works on exploiting acoustic susceptibility of MEMS sensors, and compares them with our work. Michalevsky et al. [MBN14] showed that gyroscopes in smartphones are sensitive to acoustic vibrations in hearable range, such as human speech, and therefore

Table 11: Summary of prior works exploiting acoustic susceptibility of MEMS sensors.

	Work	Attack scenario	Sensor	Setup[a]	Distance
Other	Michalevsky et al. [MBN14]	Recognizing speech with gyroscopes as microphones	gyr.	T: consumer speaker; R: smartphone	–[b]
	Trippel et al. [Tri+17]	Disturbing and controlling accelerometer output.	acc.	T: consumer speaker; R: acceler.-equipped device	–[c]
	Son et al. [Son+15]	Disorienting drones by disturbing gyroscope output	gyr.	T: consumer speaker; R: drone flight controller	≈17cm (achieved); ≈37m (expected)[d]
Covert channels	Farshteindiker et al. [Far+16]	Exfiltrating data from a surveillance implant	gyr.	T: piezoelectric transducer; R: smartphone	≈0cm (physical touch)
	Block et al. [BNN17]	Breaking Android application sandboxing	acc.	T: smartphone; R: smartphone (T=R)	≈0cm (intra-device)
	This work	Ultrasonic cross-device tracking	gyr.	T: low/high-quality speakers; R: smartphone or smartwatch	from 35cm (low SPL) to 16m (high SPL)

[a] T: transmitter; R: receiver
[b] not evaluated; smartphone is placed "as close as possible to speakers"
[c] not evaluated; sensor is located ≈10cm from the speaker
[d] potential distance when using dedicated Long Range Acoustic Devices (LRADs)

can be used as low-frequency microphones. Son et al. [Son+15] tested 15 kinds of MEMS gyroscopes against acoustic vibrations, demonstrating that gyroscope measurements can be disturbed by intentional sounds at resonant frequencies, and exploited this fact to disorient drones by affecting their gyroscopes. Trippel et al. [Tri+17] confirmed that MEMS accelerometers used in modern mobile and wearable devices are also susceptible to similar acoustic attacks.

To the best of our knowledge, utilizing MEMS gyroscopes as receivers for communication channels was proposed for the first time by Farshteindiker et al. [Far+16]. A low-powered piezoelectric transducer was considered to physically touch the surface of a smartphone (≈0cm distance) and to send data with minimal possible power. Block et al. [BNN17] proposed to exploit acoustic resonance of MEMS accelerometers to establish a covert channel between two mobile applications within one smartphone. In our work, we instead focus on transmitting data *over a distance*, with commonly-used audio hardware, and consider the practical cross-device tracking scenario with victim users naturally using their mobile devices. Finally, in a recent work [Gur21a], Mordechai Guri presented a speaker-to-gyroscope covert channel, similar to our work, with a focus on data exfiltration from air-gapped systems, and achieved comparable transmission rates and distances of up to 8 meters on three other smartphones, confirming the feasibility of gyroscope-based covert channels on modern devices.

5.3 APPLICATION SCENARIO: CROSS-DEVICE TRACKING

We consider cross-device tracking scenarios with two devices and an adversary, who is trying to link information about the victim user or their activity on one device (transmitter) with user activity or profile on another device (receiver), by transmitting a unique tracking identifier between them. The transmitter is assumed to be equipped

Table 12: Setup of different cross-device tracking scenarios.

	Web tracking	TV tracking	Location tracking
Distance	short (10–50cm)	medium (0.5–3m)	long (>1m)
Speaker quality	low	medium	high
Sound level[a]	low (≈60–70dB)	medium (≈70–85dB)	high (⩾85dB)
Transmitter device belongs to	victim	victim	attacker
Special requirements	in-browser implementation	background noise	device movements

[a] measured near the source

with a non-muted speaker, while any gyroscope-equipped smartphone or a smartwatch is considered as the receiver.

The transmitter encodes the tracking identifier into ultrasonic sounds and plays them through the speaker. We assume that the attacker has control over an application or a webpage on the receiver, which records gyroscope data, captures transmitted signals and decodes the ID. This malicious application or a web page does not require any user permissions, unlike ultrasonic tracking implementations which rely on access to the microphone. Therefore, in our case code can be hidden in any application which the user is likely to install, or can be embedded on any webpage. To successfully capture ultrasonic signals, the receiver device is assumed to be naturally located near the transmitter. The actual achievable distance is evaluated in Section 5.5.3 for different use cases.

Following existing research works [Arp+17; Mav+17] and commercial implementations (Shopkick, Lisnr, Signal360 and Silverpush) with microphones as receivers, we consider three real-world applications of uXDT, summarized in the Table 12.

WEB TRACKING. In this scenario, the victim visits a web page, which aims to track or de-anonymize the browsing session. We assume that the web session is protected from traditional tracking mechanisms, such as tracking cookies or browser fingerprinting (e.g., by using private browsing modes, disallowing cookies, etc.). We further assume that the victim has a malicious application on another device (a smartphone or a smartwatch), placed nearby, e.g., on the same working desk. Then the attacker can transmit the tracking ID between the devices and link it to a concrete user. Furthermore, instead of requiring an installed application, it is enough to have another attacker-controlled web page opened on user's smartphone. Both transmitting and receiving web pages either belong to the attacker, or only contain attacker-controlled components, similarly to a technique of embedding third-party advertisements or analytics components. Therefore, tracking code can potentially appear on thousands of websites, which increases the scale of the attack.

For this scenario, the transmitter is assumed to have only low-quality speakers (e.g., internal speakers of laptops and smartphones) and a comparably low volume level. In this setup, we believe that even a short distance between devices (10–50cm) is practical to make cross-device tracking approach applicable. We evaluate the transmission distance of our approach in Section 5.5.3, and investigate how transmitter and receiver can be implemented on web pages in Section 5.5.5.

TV TRACKING. In this scenario, the adversarial TV media provider embeds ultrasonic beacons with encoded tracking IDs into broadcasted TV content. By capturing these IDs with an application installed on user's mobile device placed nearby, the adversary can track what and when users watch. In comparison to the web-tracking scenario, in this scenario we assume that TV systems contain higher-quality audio hardware, and volume level is usually higher, around 75 dB at a source [Com20]. However, in this case we must take into account that ultrasonic signals will not be played individually, but rather embedded into existing TV audio content. In Section 5.5.6, we evaluate how robust is the transmission in presence of background noise.

LOCATION TRACKING. In this scenario, the attacker places ultrasonic beacons at specific locations (e.g., in shops) and captures tracking IDs by a malicious application on victim's smartphone. This way, the captured ID reveals the user location. Unlike other considered scenarios, in this case the transmitter is fully under attacker's control. Therefore, we assume the transmitter to have high-quality audio hardware, and the signal to be emitted at a maximum possible volume. However, in this case the signal must be captured at higher distances (at least 1–3m), and a receiver is unlikely to remain static, due to movements of the device. We evaluate robustness of our solution against natural device movements in Section 5.5.7.

5.4 COVERT CHANNEL DESIGN

The easiest way to encode tracking IDs into ultrasonic signals is to apply on-off keying (OOK) modulation: generate and play a sine wave at the resonance frequency to encode a 1, and produce no sound to encode a 0. By observing resulting gyroscope disturbances within time frames, the binary data can be decoded. In particular, this encoding has been applied to use MEMS sensors as receivers for ultrasonic covert channels at zero distance [Far+16] and within the same device [BNN17].

In this work, we propose a more advanced modulation scheme, which allows us to transmit binary data at a larger distance, potentially at higher rates, and apply the solution to practical scenarios. Our method is based on the fact that the resonant signal causes the gyroscope sensing mass to *vibrate*, i.e., a strong signal power becomes noticeable at a specific frequency (further referred to as *resulting frequency*) in gyroscope measurements. Moreover, we discovered that sounds played at frequencies slightly different from the resonant one ($\pm$10 Hz), subsequently cause different values of resulting frequencies in gyroscope measurements. Figure 19 (a) shows recorded gyroscope measurements for a stationary smartphone located near the speaker, after playing consecutively four 2-second sine waves with 5 Hz step, starting from a resonance frequency, together with a signal spectrogram. The resulting frequency components remain noticeable even for disturbance caused by signals of lower sound pressure level, when actual disturbance of measurements becomes indistinguishable from a background noise (Figure 19 (b)), or in presence of disturbances caused by device movements, e.g., when the device is held in hand (Figure 19 (c)). Therefore, in our implementation we utilize several frequencies around the resonance frequency, and observe the spectral characteristics of the signal.

More specifically, to transmit binary data, we empirically choose N frequencies close to the resonance frequency (further referred to as *transmitting frequencies*), and apply

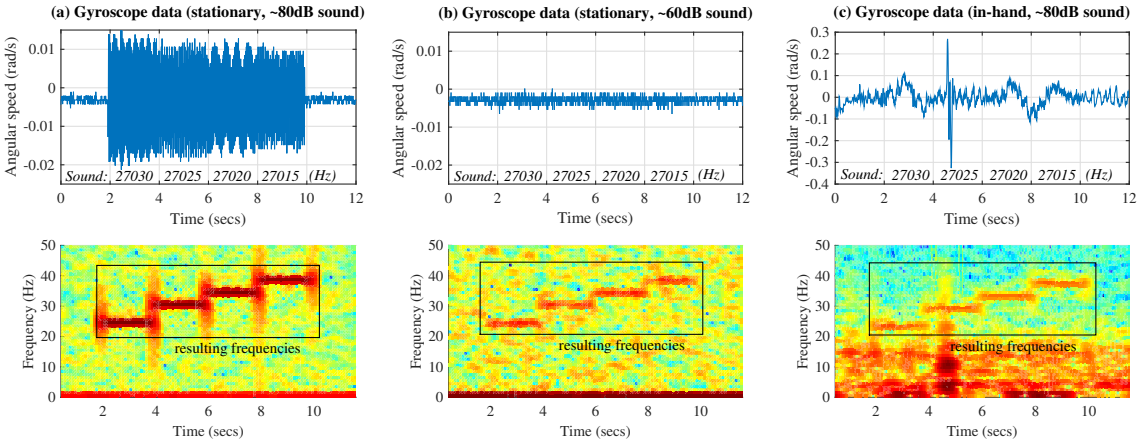


Figure 19: Example of gyroscope disturbance: gyroscope measurements recorded while playing 4 sine waves of near-resonance frequencies to an iPhone 6s in a stationary position within 50cm from a speaker at 80dB (a), at 60dB (b), and when it is held in hand (c), together with corresponding spectrograms.

a Multiple Frequency Shift Keying (MFSK) modulation scheme: each of N frequencies is associated with a binary sequence of $\log_2 N$ bits, so that a sine wave of a particular frequency, played within a time frame, encodes the represented sequence. The actual number of N is limited by the fact that the resulting disturbance of gyroscope measurements (and subsequent frequency power) gradually degrades when the frequency of sound is changing starting from the resonance frequency. One can observe such attenuation in Figure 19 (a). Moreover, for some of the transmitting frequencies we observed additional distortions and aliasing in lower frequencies of the received gyroscope signal, which reduced the range of potential frequencies to be used for transmission. In our experiments, we successfully utilized 4 frequencies, which allowed us to double the bitrate in comparison to OOK modulation.

Speakers may produce hearable audio clicks at the beginning and the end of ultrasonic transmission, due to abrupt changes of the amplitude [Des14]. To prevent them, a Hann [Opp99] window is applied at the beginning and the end of each generated sine wave. Additionally, in order to help the transmitter recognize the start of the transmission, each signal is prepended with a sequence of several short sine waves played at the resonance frequency with a small pause between them. In our experiments, we used three waves of 250ms each.

To decode the signal, the receiver first applies the short-time Fourier transform (STFT) for the resulting frequency with a window size equal to the duration of a single wave in the preamble. A signal start is detected in resulting STFT values by looking for a peak/no-peak sequence corresponding to the signal preamble. Then, the signal is decoded bit by bit. Within each time frame, a Fast Fourier transform (FFT) is calculated for all N resulting frequencies (corresponding to chosen transmitting frequencies). The binary sequence corresponding to a resulting frequency with the highest FFT-magnitude is chosen as transmitted within the time frame.

In practice, the transmitter will not know the resonance frequency of the receiver's gyroscope. To target multiple devices, the binary ID must be modulated into several sound waves, corresponding to different resonance frequencies. Then these sounds can

Table 13: Resonance frequency of gyroscopes on tested mobile devices.

Device	Gyroscope Model	Sampling Rate	Resonance Frequency
Samsung Galaxy S7[a]	STM[b] LSM6DS3	500 Hz	20.20 kHz
Samsung Galaxy S8[a]	STM[b] LSM6DI	500 Hz	20.92 kHz
iPhone 6s	IS[b] MP67B	100 Hz	27.02 kHz
LG Nexus 4	IS[b] MPU6050	200 Hz	26.90 kHz
Sony Smartwatch SWR50	Bosch BMX055	200 Hz	25.48 kHz

[a] Two devices of the same model were tested to prove identical behavior
[b] IS: InvenSense; STM: STMicroelectronics

be either played subsequently, or combined into one signal with equal weights. For simplicity, we present evaluation results considering one sine wave at a time.

5.5 EVALUATION

In this section, we first examine various mobile devices and evaluate the resonance frequencies of their gyroscopes. Then, for two smartphones, we test how the amplitude of the signal depends on the sound pressure level (SPL). Afterwards, we evaluate the proposed encoding scheme, by showing how SPL affects the bit error rate (BER), determine a transmission bitrate, and demonstrate the achievable distance. Finally, we evaluate our approach when signals are transmitted between web pages (web tracking), how robust is it against background noise (TV tracking) and device movements (location tracking).

5.5.1 *Resonance frequencies*

For our experiments, we choose four modern smartphones and a smartwatch. To detect their resonance frequencies, we place the devices directly near a speaker, and generate sine waves at frequencies from 18 kHz to 30 kHz with 20 Hz increments. For each played sound, we calculate the average magnitude of the resulting frequency, and choose the sound frequency which caused the strongest signal. In all our experiments (unless stated otherwise), we use a single external speaker KRK Rokit 5 G3 connected to a MacBook Pro A1502 laptop, configured to output sounds with 96 kHz sampling rate. All measurements are taken in a typical office environment.

Table 13 lists the tested devices, their gyroscopes, available sampling rates, and discovered resonance frequencies. All the devices in our experiment have gyroscopes susceptible to ultrasonic sounds. Gyroscopes manufactured by InvenSense resonate at 25–27 kHz, while the STMicroelectronics sensors are susceptible to 20.9–21.4 kHz sounds, which is especially interesting, since sounds in these ranges can be generated even with 44 kHz audio hardware. Although other researchers discovered the gyroscopes which do not resonate in the ultrasonic range[Son+15], we believe that the attack is practical, since very popular devices (e.g., modern Apple and Samsung smartphones) *are* affected. For further experiments, we choose two smartphones with the highest noticeable disturbance caused by sounds, namely Samsung Galaxy S7 and iPhone 6S.

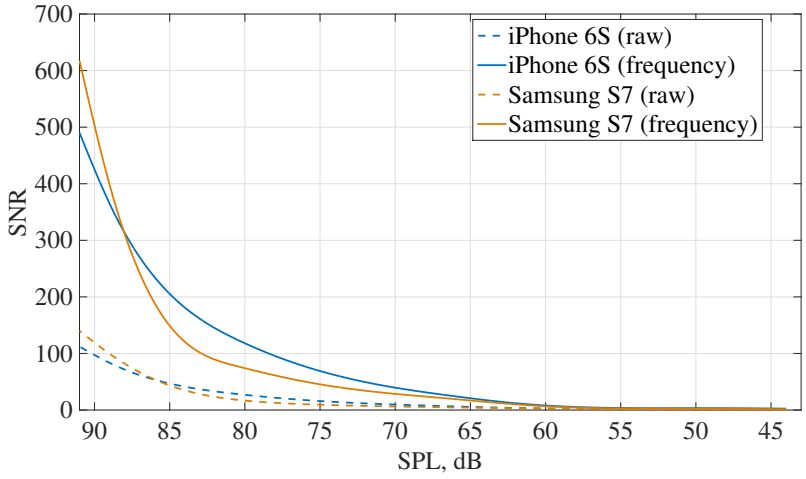


Figure 20: SNR levels recorded for two smartphones located in 50cm from a speaker, depending on the SPL, measured at the same point. A signal based on a magnitude of the resulting frequency (solid) is compared to a signal based on raw disturbance of the measurements (dashed).

5.5.2 *Signal-to-noise ratio*

In this experiment, we estimate the dependency of the signal strength on the SPL, independently of the payload and used encoding scheme. For this purpose, we gradually reduce speaker volume from the maximum (by ≈5dB) and, at each volume level, play a sine wave at the smartphone's resonance frequency. We record the produced sounds using a microphone and calculated the resulting SPL, while recording the gyroscope data on the smartphone. To precisely measure SPL for high-frequency sounds, we use a calibrated Earthworks QTC30 microphone connected to a Fireface UC sound card, with flat frequency responses of up to 30kHz. Then, we calculate the signal-to-noise ratio (SNR), i.e., the ratio between the average magnitude of the resulting frequency in FFT values for gyroscope measurements when the sound is played, and without any sound produced. For comparison, we also calculate SNR as the ratio between raw disturbances, by calculating the standard deviation of the measurements.

Figure 20 shows the resulting SNR levels. One can see that the resulting SNR is stronger for high SPL (80–90dB), and the signal is slightly stronger for the iPhone 6S. For both smartphones the SNR rapidly attenuates and becomes comparable to noise at SPL lower than 60dB. Nevertheless, one can see that the signal based on spectral characteristics is stronger in comparison to the signal based on raw disturbance.

5.5.3 *Transmission*

In this experiment, we evaluate data transmission using the proposed MFSK encoding scheme. For this purpose, we use the same experimental setup as in the previous experiment, transmit tracking IDs with different bitrates and volume levels (50 IDs of a length of 30bits in each case), and measure how SPL and the chosen bitrate affect the resulting bit error rate (BER). We consider a BER of $<10\%$ as practically suitable, since in this case error-correcting codes can be used to ensure correct decoding with manageable

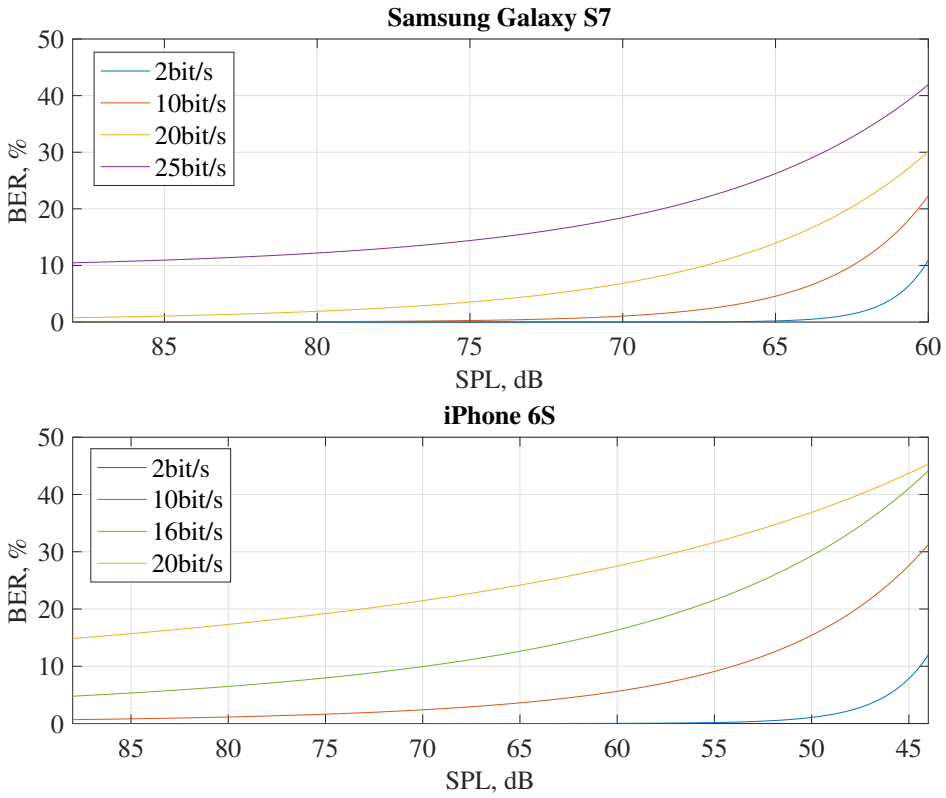


Figure 21: Dependency of BER on the SPL. Smartphones are placed in 30cm from the speaker, the sound is recorded with a measurement microphone at the same point. Average of 50 transmissions for each bitrate and SPL.

overhead. For example, a primitive narrow-sense BCH(31,16) code [RC99] can be applied to correct 3 bits of a 31-bit code (in a channel with BER of up to 9.7%), with a payload block of 16-bit.

Figure 21 shows the resulting BERs for two smartphones. One can see that results comply with the SNR experiment, with decoding errors appearing when the raw signal becomes weak. The iPhone 6S, which has a higher SNR, performs better for lower SPL values (45–60dB). The transmission bandwidth is limited by the measurement sampling rate: at some point (20bit/s for the iPhone with 100Hz sampling rate, and 25bit/s for the Samsung with 500Hz sampling rate), the number of samples within a recording time frame becomes too small to correctly identify the resulting frequency, even with a high-level SPL. Based on the results, we consider bitrates of 10bit/s for low SPLs (>62dB) and 16–20bit/s for high SPLs (>68dB) as practically suitable for corresponding tracking scenarios.

5.5.4 *Distance*

To theoretically estimate the maximum possible distance for the transmission, we use the dependency between SPL and distance from the source, known as *inverse square law*

Table 14: Maximum distance of the transmission.

Reference SPL (at 50cm)	Max. distance (10bit/s)		Max. distance (20bit/s)	
	estimated	**confirmed**	estimated	**confirmed**
70dB	1.26m	**1.2m**	0.63m	**0.6m**
80dB	3.97m	**3.5m**	1.99m	**2.0m**
95dB	22.33m	**16.0m**	11.19m	**9.0m**

[DJ89]: given the reference SPL_{ref} measured at a distance d_{ref}, the SPL depends on a distance d as

$$SPL = SPL_{ref} - 20\log(d/d_{ref}),$$
$$d = d_{ref} * 10^{|SPL_{ref} - SPL|/20}.$$

Given the last formula and boundary SPL values found in the previous experiment (62dB for 10bit/s, 68dB for 16–20bit/s), we can estimate the transmission distance depending on a reference SPL. In practice, the actual distance can be also affected by reflections, reverberations, interference, the direction of the sound wave propagation with regard to the receiver, etc. Table 14 shows the estimated and practically confirmed distances d for the reference SPLs of 70dB, 80dB, and 95dB (maximum for our speaker) at $d_{ref} = 50$cm. All measurements are taken on a Samsung Galaxy S7 smartphone, in a typical office room (up to 3.5m) and in the office corridor (>3.5m). As one can see from the table, our practical results comply with theoretical estimation. When the source SPL is high (⩾80dB at 0.5m), a distance of 3.5–16m is achieved, which proves applicability of our approach to TV and location tracking scenarios.

To precisely evaluate the distance when the source SPL is low and the transmitter is equipped with comparably low-quality speakers (e.g., in the web-tracking scenario), we transmit tracking IDs on a laptop using only internal speakers, and measure the area around the laptop, where the signal was successfully decoded (BER<10%), depending on a system volume level. The results are shown in Figure 22. The resulting distance is limited to 45cm, but even this area can be sufficient to transmit a tracking ID to a smartphone naturally located near the laptop on a working desk. We consider a volume level of 75% and corresponding distance of up to 35cm as realistic.

5.5.5 *Web tracking: in-browser implementation*

To confirm applicability of the approach to the web-tracking scenario, we implement web-based versions of both transmitter and receiver. We use the Web Audio API to play sounds on a web page, and confirm that it works in desktop and mobile versions of modern browsers (Chrome, Firefox, Tor Browser, and Safari). We must note that the signal transmission is not completely covert to users: in all *desktop* browsers, a small notification icon appears when the tab is playing sound. Moreover, some browsers prevent automatic playback of audio without an explicit user interaction. Nevertheless, the signal can be embedded into benign audio or video content, not being suspicious to users. We also discover that Safari does not show the notification icon if sound is played for less than 1.5s, enabling completely covert transmission in a short time.

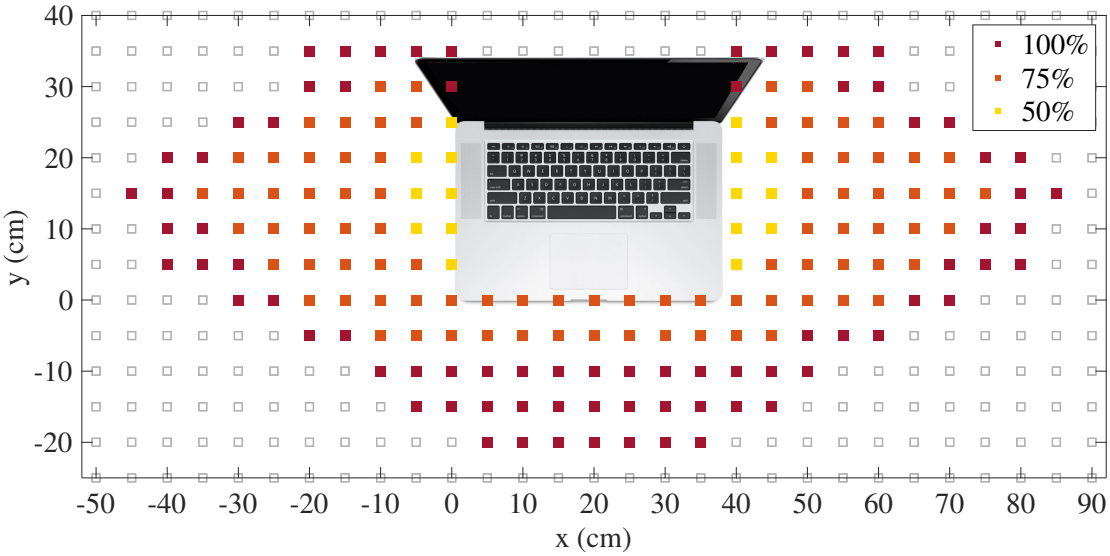


Figure 22: Area around laptops where the signal can be decoded with BER<10%, depending on a volume level set using default system volume control (50%, 75%, and 100% of the maximum volume level).

To record the gyroscope measurements from a web page, we use the DeviceMotion API. This API does not provide *raw* gyroscope data, but instead uses a combined signal from several sensors (so-called *sensor fusion*), which reduced the resulting disturbance of the measurements. Moreover, the sampling rate is reduced to 60–100Hz, depending on the browser. As a result, we are able to achieve only 5bit/s bandwidth in mobile browsers. On the Chrome mobile browser, the amplitude of the combined signal is significantly lower in comparison to raw gyroscope data, so we are able to decode the signal only in close proximity to the source (≈80dB).

5.5.6 *TV tracking: robustness against noise*

To prove the robustness of the transmission against background noise, we transmit tracking IDs with 80dB sounds in presence of background sounds, played in parallel through another speaker: background music and TV news broadcasting (≈80dB), and white noise (60, 80, and 95dB). The signals are captured on the Samsung Galaxy S7 smartphone located at 3m distance. We find that the resulting decoding is not affected at all ($\leqslant$1% increase in BER), not only by music and news broadcasting (which is expected, since these sounds mostly lie within 20Hz–20kHz range), but even by white noise significantly exceeding the source signal (95dB). Therefore, gyroscopes naturally filter acoustic noise of other frequencies, making the sensor a robust receiver for tracking signals.

5.5.7 *Location tracking: robustness against movements*

To test how robust the proposed method is against device movements, we transmit tracking IDs using a signal with SPL of 95dB (measured at 50cm distance), and capture the signal on the Samsung Galaxy S7 (at ≈3m distance), under the following conditions:

Table 15: Robustness of the covert channel against movements.

Experiment	BER (20bit/s)	BER (10bit/s)
Static position (no movements)	2%	1%
Smartphone in the hand	7%	4%
Smartphone is used	11%	5%
Person is walking	28%	22%

the smartphone is held in a hand, the smartphone is used to play a simple game, and the person holding the smartphone is freely walking near the speaker. In each case, we transmit 50 IDs with a bitrate of 20bit/s and calculate the resulting BER.

The results are presented in the Table 15. The transmission remains stable in presence of slight movements: when the smartphone is held in a hand, or is being naturally used at the time of recording. When the person is actively moving, however, additional errors appear. Nevertheless, we believe that the approach is applicable to the location tracking scenario at specific locations, where the person is not *always* moving (e.g., in shops), since the comparably high bandwidth allows to quickly transmit the tracking ID.

5.6 COUNTERMEASURES

Several ways to prevent the presented cross-device tracking are possible:

- Similarly to countermeasures against reaction of magnetometers to electromagnetic interference, physical shielding of gyroscopes is the most straightforward way to limit acoustic susceptibility of the sensor. Existing experiments show that a layer of foam, paper or aluminum reduces the disturbance of the sensor by 16–60% [Son+15]. However, we expect that this measure mismatches with an industry trend of making consumer mobile devices thinner and lighter.

- Designing the gyroscopes to have a resonance frequency of at least more than 25kHz prevents the attack from being feasible, at least with audio hardware that does not support high-resolution sampling.

- A low-pass filter can be applied to gyroscope measurements on the hardware or OS level to prevent detection of the transmitting frequency. However, precise gyroscope measurements can be required for some apps.

- Mobile OS vendors can forbid access to gyroscope data without an explicit permission granted by the user. However, users may not correctly estimate potential privacy threats [Meh+18]. Moreover, most of mobile devices in use run outdated operating system versions [Dis18] and do not regularly receive security updates [Mos20].

- Starting from version 8.0, Android allows applications to run in a background only for a limited period of time, similarly to iOS. We advocate this measure and believe that additional restrictions can be introduced for access to sensors from background processes.

Most described countermeasures would require hardware or software changes, and may have performance or production cost drawbacks. Therefore, the presented attack remains completely feasible at present and can cause significant privacy threat to end users.

5.7 SUMMARY

In this chapter, we presented a novel approach to establish ultrasonic cross-device tracking that utilizes susceptibility of gyroscopes in modern mobile devices to acoustic resonance. We showed that observing the reaction of gyroscopes to resonance frequencies in the signal spectrum allows us to reliably capture tracking signals at a distance, making the approach applicable to web tracking, TV tracking and location tracking scenarios. Although the achieved distances and transmission rates are smaller in comparison to the approach of using microphones to receive the signals, the presented method does not require any explicit permissions and can be run unnoticed to end users.

6

SENSOR-BASED VIBRATIONAL COVERT CHANNEL

In this chapter, we introduce a covert channel that utilizes susceptibility of accelerometer sensors in smartphones to vibrations caused by the loudspeaker's woofer producing low-frequency sounds. This effect allows an application on a smartphone to capture acoustic signals inaudible to human ears at a distance. Unlike microphones, access to accelerometers does not require explicit user permissions. Therefore, the exploited reaction can be exploited in scenarios introduced in Chapter 5 and to establish *zero-permission* cross-device tracking.

Remarks. Content in this chapter is based on the corresponding publication [Mat+19].

6.1 MOTIVATION AND CONTRIBUTIONS

Utilizing inaudible acoustic signals as a medium for covert communication channels has been widely discussed in previous research, for a survey see [CA16]. Typically, ultrasonic sound (above 18kHz) is used for such a communication, as it makes transmission inaudible to humans, once the average hearing sensitivity of ≈20Hz–20kHz [Lee+12] is exceeded. Ultrasonic covert channels have been applied to a variety of attacks and applications, including data exfiltration from air-gapped systems [HG14; CA14; Des14] and sandboxed applications [BNN17], and cross-device tracking [Arp+17; Mav+17].

To limit privacy threats emerging from acoustic covert channels, modern operating systems require explicit user permissions to access the device's microphone [Per18]. However, as we demonstrate in Chapter 5, gyroscope sensors in smartphones can be used as zero-permission receivers of ultrasonic covert channels. The applicability of ultrasonic acoustic covert channels is, however, limited by the capabilities of the audio hardware: The resonance frequency of most MEMS gyroscopes lies above 25kHz [Son+15; MSK18], while sound processors in computers usually have a default sampling rate of 44.1kHz or 48kHz, and therefore cannot produce frequencies above 24kHz without aliasing.

In this chapter, we explore an alternative way to establish acoustic covert channels. Instead of using ultrasonic frequencies, we propose to use frequencies *below* the audible range, down to 16–24Hz. The hearing sensitivity falls sharply as the frequency goes below 50 Hz, i.e., humans can perceive such low frequencies only at a high sound pressure level (SPL) [WCR72]. At the same time, speakers are normally designed to accurately reproduce signals only in the audible range, which often results in noticeable attenuation of the signal energy at frequencies in the lower range. As a result, at practical volume levels, set to conveniently perceive audible music or speech, the signal energy at low frequencies can be much lower compared to the audible range. As a result, as we show in our evaluation, signals deliberately produced at these frequencies remain inaudible to humans, but can be still captured using a microphone.

Furthermore, when a speaker produces low-frequency signals, the movements of the woofer's membrane lead to slight vibrations of the speaker enclosure, which propagate further to the surface where the speaker is located (e.g., a table). We have found that

these slight vibrations, imperceptible to humans, can be successfully captured by the accelerometer sensor of a smartphone lying on the same surface. In this manner, the accelerometer can act as a zero-permission receiver of the *vibrational* covert channel, with the vibrations being deliberately produced by the speaker when playing inaudible low-frequency sounds.

We evaluated the produced vibrations for three speakers, and implemented the transmission, achieving a raw bitrate of up to 5bit/s with the bit error rate below 10%. Although bandwidth is limited in comparison to ultrasonic covert channels and transmitter and receiver must share the same surface, the proposed approach does not require high-resolution audio support for the transmitter or user permissions for the receiver, which increases the applicability of the covert channel.

CONTRIBUTIONS. Our contributions are the following:

- We show that woofer-equipped speakers can often produce sounds at frequencies as low as 16–24Hz, which can be inaudible to human ears. More importantly, when playing these sounds, woofers produce slight vibrations of the surface on which the speaker is located, which can be unperceivable to humans, but detectable by accelerometers on mobile devices.
- We propose to utilize this effect to establish a zero-permission vibrational covert channel. We identify the range of frequencies at practical SPLs which can be used for such a covert channel, and implement the transmission.
- We evaluate the proposed approach for several consumer speakers, and discuss several potential application scenarios and possible countermeasures.

6.2 BACKGROUND

In this section, we briefly describe use of MEMS accelerometers in mobile devices, their susceptibility to acoustic vibrations, and abilities of modern audio hardware to transmit covert acoustic signals.

6.2.1 *Audibility of low-frequency sounds*

It is traditionally considered that the human hearing limit lies within the so-called *audible* range of ≈20Hz–20kHz. In fact, the hearing sensitivity of human ears depends on the sound frequency and the sound pressure level (SPL): For each particular frequency, there exists a minimum SPL threshold required to perceive the sound [ISO22603]. In particular, humans can hear frequencies below 20Hz, if the SPL is sufficiently high. However, this threshold is significantly higher at the edges and outside the audible range. For example, the average SPL threshold for the frequencies of 25 Hz and 12.5 Hz are ≈65dB and ≈95dB[1], respectively [WM90; WCR72]. For comparison, the hearing threshold for mid-range frequencies of 400Hz–4kHz is below 10dB [ISO22603].

In our work, we rely on the difference between hearing sensitivity for different frequencies: users may set the "loudness" of their speaker to a level which is convenient to

1 Throughout the text, the reference sound pressure for SPL in decibels is $2 * 10^{-5}$Pa.

hear audible frequencies (e.g., 70dB SPL for 1kHz); however, the signal produced at a low frequency (e.g., 25 Hz) would be inaudible as it is below the hearing threshold.

6.2.2 *Speaker low-frequency capabilities*

The primary function of a loudspeaker is to convert incoming electrical frequency signals into actual sound waves by performing physical vibrations. A detailed explanation of a speaker's design can be found in [Tal12; Dic05]. In summary, a loudspeaker consists of one or multiple *drivers*, responsible to produce frequencies of a specific range. The drivers producing low frequencies are called *woofers* or *subwoofers*, where the latter usually produces lower frequencies (we use these two terms interchangeably). The most common *moving-coil driver* consists of a lightweight diaphragm connected to an electromagnet called *voice coil*, located within a permanent magnetic field. Applying a current produces electromagnetic force due to the interaction between magnetic fields, which causes the coil and diaphragm to oscillate back and forth, generating air pressure. The coil and diaphragm are connected to the speaker's enclosure through a spring called *spider*, which returns the diaphragm to its neutral position. The vibrating diaphragm, anchored to the enclosure through the spider, inevitably causes the enclosure itself to vibrate. As we observe in our experiments, these vibrations can further propagate to the surface on which the speaker is located and be measured by accelerometer sensors.

An important characteristic of the speaker is its frequency response, which describes the difference in produced SPL for the same input gain, depending on a frequency [Tal12]. Usually, the speaker has an operational frequency range with comparably "flat" frequency response. Outside this range, however, the signal quickly attenuates. This fact further increases the range of frequencies which may be practically inaudible to human ears: For example, a speaker producing 90dB SPL at 1kHz frequency may produce only 60dB for 25Hz frequency, below the hearing sensitivity threshold, making the resulting signal inaudible.

6.2.3 *Susceptibility of accelerometers to low-frequency sounds*

Most modern mobile devices are equipped with accelerometers, which measure the external acceleration force that is applied to the sensor along three physical axes, including the force of gravity, measured in m/s^2. The sensor data allows identifying and measuring device motions, such as tilt or shaking, and is used in games, virtual reality applications, etc. In Android, iOS, and web applications, 3-axis accelerometer measurements can be retrieved by using the Sensor API [Sen18], Core Motion [App18b] framework, and *DeviceMotion* event [Dev18], respectively. The sampling rate of the measurements ranges between 60–500Hz, depending on the sensor and device.

In our work, we show that if the mobile device is located on the same surface as a loudspeaker, its accelerometer is sensitive enough to capture the slight vibrations of the reference surface itself, caused by the loudspeaker.

6.2.4 *Related work*

Deshotels [Des14] established a vibrational covert channel on mobile devices, by using the vibration motor of a smartphone as the transmitter. The idea of using accelerometers to capture vibrations from the speakers was introduced by Hasan et al. [Has+13]. Authors hypothesized that low-frequency audio signals produced with strong subwoofer system could be inaudible and detectable using accelerometers at a distance of a few feet, but did not evaluate this idea. In our work, we show that the signal can actually be produced by comparably small consumer speakers with a woofer, demonstrate how slight vibrations produced by sounds of low SPLs can be effectively decoded, and evaluate the transmission by identifying usable ranges for frequencies and SPLs.

6.3 COVERT CHANNEL DESIGN

In this section, we describe the attack scenario, analyze how the vibration is presented in accelerometer data, describe the encoding scheme, transmission details, and methods to receive the signal.

We consider a generic covert channel attack model with two devices and an adversary, who is trying to extract sensitive information from one device (transmitter) by sending it over the covert channel to another device (receiver). The transmitter is assumed to be connected to a speaker equipped with a woofer. Any accelerometer-equipped mobile device (e.g., a smartphone) is considered as receiver. We further assume that transmitter and receiver are sharing a common surface, e.g., a wooden office table.

On the side of the transmitter, the attacker is able to encode the information to be exfiltrated into low-frequency sounds and play them through the speaker. We consider the case that the attacker may not be able to control the volume of the speaker, and instead has to play the signals at a level set by the device user.

On the receiver, the attacker has control over an application or a web page which records accelerometer data, captures the transmitted signals and extracts the information. Unlike traditional implementations of acoustic covert channels, which rely on access to the microphone, in our case the receiving code does not require any user permissions, except for access to the Internet, granted by default on modern operating systems. Therefore, it can be implemented as a malware, hidden in any application or a web page which the users are likely to open on their device. Finally, the decoded information is sent to the attacker over the Internet.

The ability to receive acoustic signals on any smartphone using just the accelerometer, without any user permissions, makes the channel applicable to a number of scenarios where a small amount of data needs to be transmitted to a device nearby. First, the covert channel can be used to exfiltrate information from so-called air-gapped systems, the scenario we discussed in Chapter 2. In our case, a malicious insider can place the smartphone in close proximity to the device's speaker, or infect a smartphone of the device user with recording malware. Second, the covert channel can be applied to the ultrasonic cross-device tracking scenario we introduced in Chapter 5, and be used to link user profiles or activities on two devices. For example, an application or a web page on the user's laptop can encode a tracking identifier into sound signals and transmit it over the covert channel, while an application on user's smartphone can regularly record

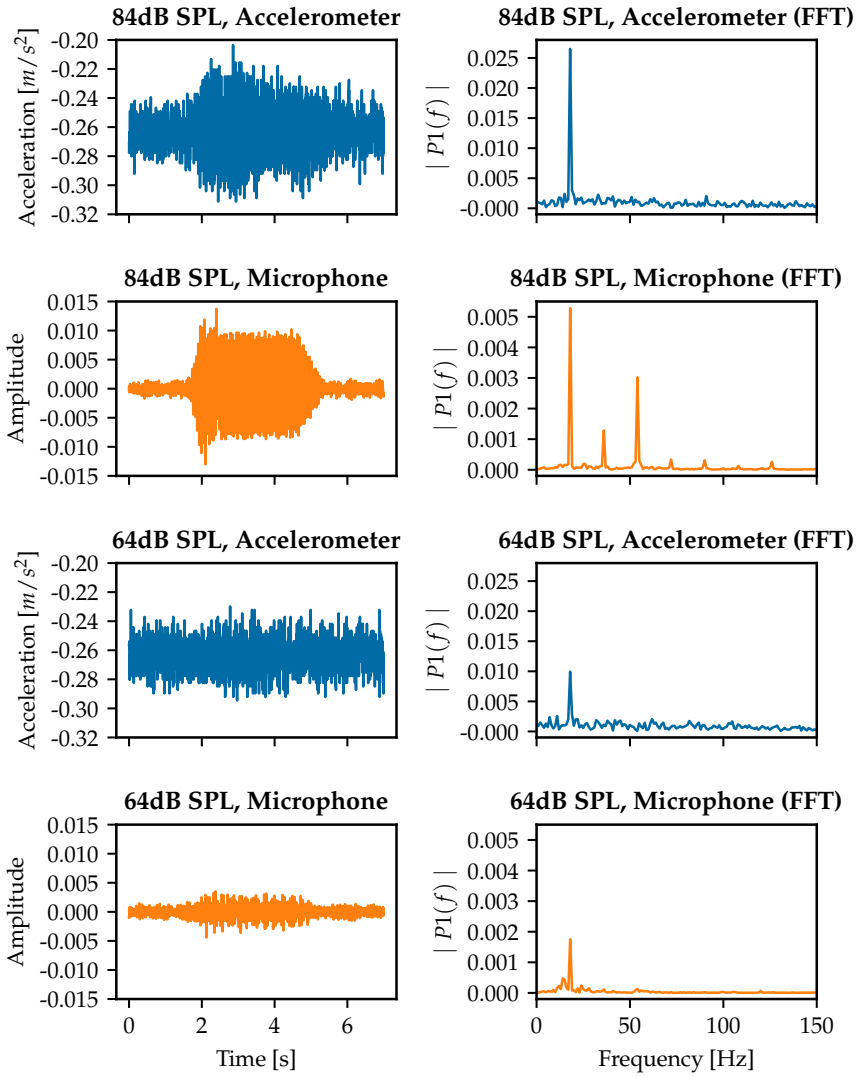


Figure 23: Example of the recorded vibrations. Accelerometer measurements (blue) and sound (orange) were recorded while playing an 18Hz sine wave at 84dB and 64dB SPL in 50cm between devices. The figure shows the raw signal and its FFT representation. A clear peak at 18Hz is clearly distinguishable in the FFT values even for the low SPL.

accelerometer data. If the recording application was able to receive the tracking ID, both devices must be located on a common surface, and are likely to belong to the same user. Finally, the covert channel can be used to exfiltrate data from isolated environments within one mobile device. In particular, it can be applicable to the web-tracking scenario we discussed in detail in Chapter 3.

6.3.1 *Analysis of the signal spectrum*

The basic idea of our covert channel is to analyze surface vibrations produced by the speaker, by observing corresponding disturbances in accelerometer sensor data. Figure 23 shows an example of such a disturbance, as well as the experimental setup. Raw accelerometer measurements recorded on a smartphone, after playing an 18Hz sine

wave on a speaker at two different SPLs. Additionally, the audio signal was recorded using a microphone located on another table at a distance of 50cm from the speaker. The figure shows raw measurements as well as their frequency domain representations after computing the Fast Fourier Transform (FFT).

The recordings were unperceivable to human ears, but the microphone was able to record the signal for both SPLs. Furthermore, when the sound was played at a high SPL, a clear disturbance in the raw accelerometer measurements can be noticed, indicating surface vibrations. The amplitude of the produced disturbance depends on the acoustic signal SPL: The woofer's diaphragm moves at larger amplitudes for the larger SPL, which results in stronger resulting vibrations of the speaker enclosure and the surface. As one can see, for the lower SPL, the raw disturbance is no longer noticeable.

For this reason, instead of analyzing the raw amplitude of the measurements, we utilize information in the frequency domain: For measurements corresponding to both SPL levels, one can see a clear peak in the signal spectrum at 18Hz, indicating that vibrations occur at the same frequency as the original sound signal. In practice, the frequency corresponding to the highest peak in the FFT can be different depending on the speaker design, surface and the relative position of the devices. For example, the speaker's diaphragm can produce vibrations for both forward and backward movements, which results in the resulting vibration frequency equal to the double value of the original sound frequency. Nevertheless, analyzing the actual frequency of produced vibrations instead of their amplitude allows us to detect even slight vibrations caused by sounds with comparably low SPL. We evaluate this approach to measure the produced vibrations in Section 6.4.2.

6.3.2 *Encoding and transmission*

To encode a binary string into sound signals, we apply on-off keying (OOK) modulation. The transmitter plays a sine wave at a chosen frequency within a time interval to encode a 1, and does not play any sound within an interval to encode a 0. This encoding scheme allows us to evaluate the basic feasibility of the covert channel; we leave the evaluation of other encoding schemes for future work.

Similarly to our approach in Chapter 5, in order prevent hearable audio clicks at the beginning and at the end of each produced since wave, we applied a Hanning window [Opp99] at the beginning and the end of each generated sine wave. However, we found that on some speakers audible clicks could not be fully avoided on high SPLs even when the window is applied. We evaluate the audibility of the transmission in Section 6.4.3.

In order to help the receiver recognize the start of the transmission within a continuous recording, the transmitted string is additionally prepended with a predefined binary sequence. We used time periods of different lengths for transmitting bits of this synchronization sequence and bits of the actual payloads, in order to avoid false positive detections when the synchronization sequence happens to appear in the payload itself. In our implementation, we used the 11-bit Barker sequence for synchronization [Bar53], due to its low autocorrelation properties, which facilitates its accurate detection.

Table 16: Frequency responses of the speakers, measured as difference in dB with SPL when playing 1kHz reference sine.

Speaker	16Hz	18Hz	20Hz	22Hz	24Hz
Apple HomePod	−26.3	−21.4	−17.2	−14.6	−12.2
iTeufel Air Blue	−29.7	−27.8	−21.6	−18.1	−16.8
KRK Rokit 5 G3	−35.6	−32.3	−30.8	−30.6	−30.5

6.3.3 *Receiving and decoding*

The receiver first converts the raw 3-axis accelerometer traces into a one-dimensional trace. We have observed that the actual direction of produced vibrations depends on the speaker design (e.g., whether the diaphragm is moving vertically or horizontally). In order to identify the direction of the vibration, we apply Principal Component Analysis [Hot33] to the data, and choose the first component as the result, as it represents the new axis with the highest data variance.

Afterwards, the receiver identifies the start of the signal. If the actual frequency of vibrations is known, the receiver applies the Short-Time Fourier Transform (STFT) for this frequency with a short window and calculates the cross-correlation between resulting STFT amplitudes and the synchronization sequence, resampled to have the same sampling rate as the sensor. A time point corresponding to a high peak in the cross-correlation is considered as the start of a transmission. If the frequency of vibrations is not known, the receiver can perform this procedure for several candidate frequencies, e.g., checking the original sound frequency and its double value, or choosing frequencies with high average energy within the recorded interval.

Using the synchronization sequence, the receiver calculates the average FFT magnitude at the frequency of vibrations within time frames corresponding to a logical 0, and, similarly, the magnitude of all measurements corresponding to a logical 1. The mean of these two averages is used as a threshold. Finally, the transmitted payload is decoded bit by bit, by comparing this threshold with the FFT magnitude for each corresponding interval.

6.4 EVALUATION

In this section, we first describe the experimental setup and measure the frequency response of the speakers. Then, for each speaker, we evaluate produced vibrations and the audibility of the transmission. Finally, for the chosen setup, we evaluate the optimal bitrate.

6.4.1 *Experimental setup*

For our experiments, we choose 3 consumer speakers equipped with a woofer, listed in Table 16. We use a Galaxy S8 smartphone as the receiver, with its accelerometer sampled at the 500Hz rate. To precisely measure SPL, we use a professional calibrated Earthworks QTC30 microphone with a flat frequency response down to 3Hz, connected to a Fireface

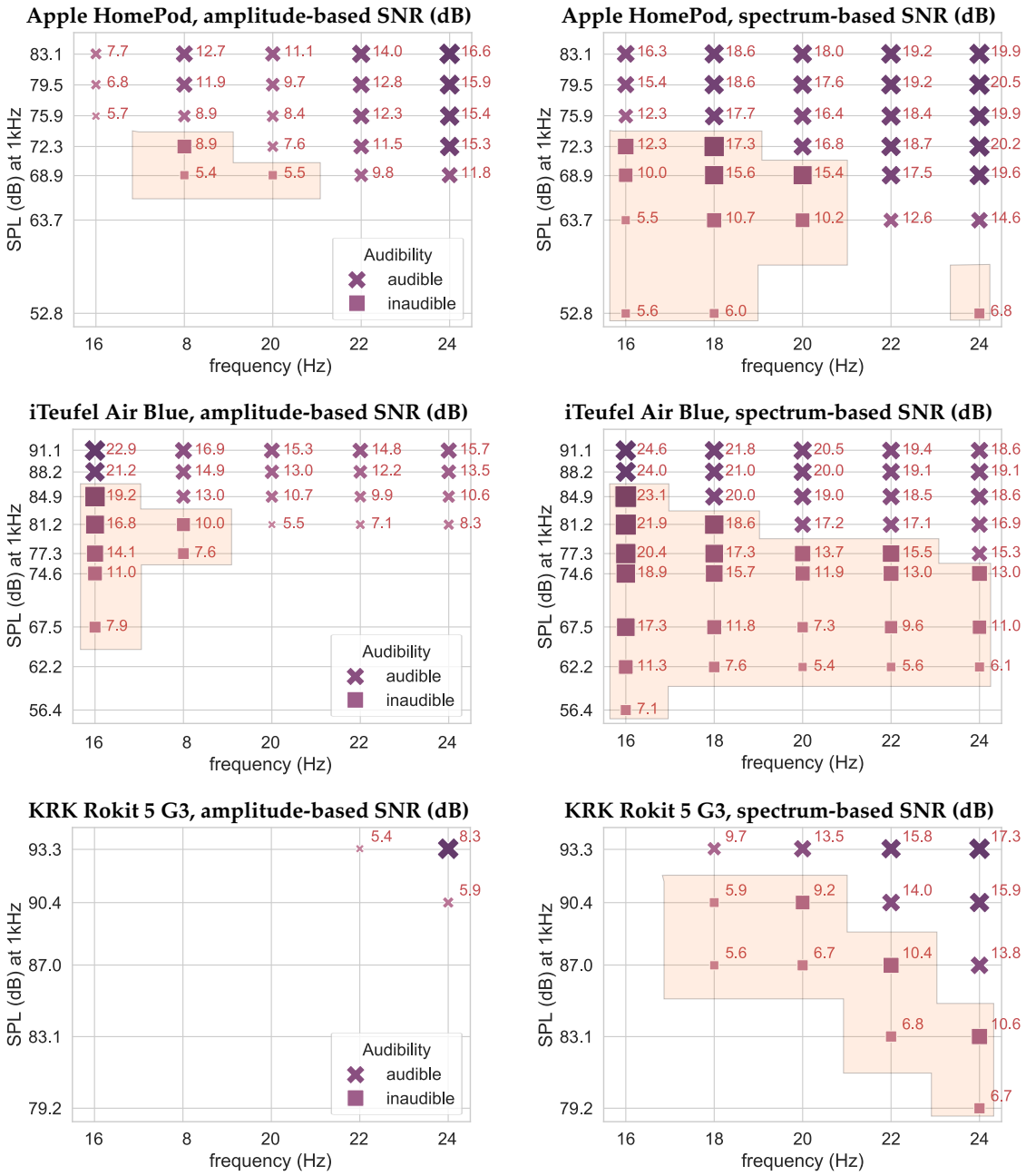


Figure 24: SNR levels of vibrations recorded on a smartphone in 50cm from the speaker, for three speakers, depending on the frequency and SPL (Section 6.4.2), together with the audibility of the frequency-SPL points (Section 6.4.3), inaudible: ■, audible: ×. An amplitude-based SNR (up) is compared to a spectrum-based SNR (down). Orange area highlights admissible range of frequencies/SPLs.

UC sound card. If not stated otherwise, all experiments are performed in a typical office environment with moquette floors; speakers and smartphones are placed on a wooden office table. The experimental setup is shown in Figure 25.

As a first step, we identify the frequency response of the speakers at low frequencies. For this purpose, we play sine waves from 24 Hz down to 16 Hz, as well as a reference sine wave at 1kHz, and measure the produced SPL at 50cm. We set the volume level

Figure 25: Example of the experimental setup.

of the speakers so that the reference 1kHz wave is played at 80dB SPL. Afterwards, we calculate the difference between SPLs produced for each frequency within 16–24Hz and SPL at 1kHz. The average results are shown in Table 16. As one can see, all three speakers can play frequencies down to 16 Hz, despite the operational frequency ranges stated in technical specifications of the speakers. However, the produced SPL at these frequencies drops significantly. This estimation shows that at practical SPLs, used to conveniently listen to the audible range sounds (e.g., 70dB for 1Khz sounds), the SPL produced when playing low-frequency sounds can fall far below the hearing threshold, making the resulting signal inaudible.

6.4.2 *Vibration signal*

Although the tested speakers can produce inaudible low-frequency sounds, they also need to conduct a sufficient amount of vibrations to the surface, to be captured by the accelerometer. In this experiment, we measure the strength of the produced vibrations.

For this purpose, a speaker and a smartphone are placed on the same table at a distance of 50cm from each other. We record accelerometer data when playing sine waves at each frequency within 16–24Hz range at different volume levels. Afterwards, we measure the signal-to-noise ratio (SNR) of the recorded signals, expressed in decibels, in two different ways. First, we measure the SNR based on the raw disturbance of the vibration, i.e., the ratio of the signal power when the sine was played, to the signal power when no sound was produced. Second, we compute the spectrum-based SNR, by applying FFT to the signal and measuring SNR as the ratio between the peak value at the vibrating frequency to the median FFT value. The latter approach corresponds to the method for detecting the signal described in Section 6.3.3.

Figure 24 shows the results for 3 speakers depending on the frequency and SPL. For readability, we include only points with sufficient SNR$\geqslant$5dB. We show SPL values produced for the reference 1kHz sine at the corresponding volume level, to indicate SNR dependency on SPL for audible sounds. One can estimate the actual produced SPLs at a particular frequency using Table 16.

As one can see, all three speakers generate vibrations which can be captured by the accelerometer. The Apple HomePod generates the strongest signal, while The Rokit 5 G3 could produce noticeable vibrations only at high SPL levels. Generally, the signal is higher for the higher SPL. Interestingly, the same trend does apply to frequencies: two speakers generate a higher signal for 16–18Hz signals in comparison to 20–20Hz. Even though the sound SPL gradually attenuates when lowering the frequency (see Table 16), the speaker may produce stronger vibrations at lower frequencies. More importantly, the spectrum-based approach significantly extends the range of frequencies and reduces the thresholds for SPL required for producing vibrations, which confirms its practical applicability.

6.4.3 *Audibility*

The previous experiment demonstrates the minimum SPL required for a particular frequency and a speaker to produce noticeable vibrations. The upper limit for the SPL is set by the hearing threshold for humans at a particular frequency. Furthermore, as we mentioned, speakers may introduce audible clicks and distortions, especially at a high SPL, even if the signal itself cannot be heard. In this experiment, we evaluate the audibility of the produced signals on our tested speakers. For this reason, we verify if randomly played 2-second sine waves of 14–40Hz at different SPLs can be heard at 1m distance in a typical office environment. We include the results for different SPLs and frequencies in Figure 24, marking each pair of frequency and SPL as audible or inaudible. As one can see, signals at peak SPLs are perceivable even for very low frequencies, either reaching the hearing threshold or due to audible clicks and distortions. However, when the SPL is sufficiently low for a particular frequency, the signal becomes inaudible.

Based on the results of the SNR and audibility experiments, we identify the admissible range of frequencies and SPLs, for which transmission generates a sufficient level of vibrations but remains inaudible. We highlight this range in the Figure 24 for each speaker (orange area). We consider the frequency range of 18–20Hz with SPL levels of ≈65–75dB as the most practical, showing good performance for Apple HomePod and iTeufel Air Blue.

6.4.4 *Transmission*

In this section, we evaluate the optimal speed of transmission when using the on-off modulation scheme and decoding method described in Section 6.3.3. For this purpose, we choose 3 sets of frequency/SPL within the admissible ranges identified in previous experiments, namely (18Hz, 65dB), (18Hz, 72dB), (16Hz, 65dB), and (16Hz, 72dB). Using each set, we transmit binary messages with different bitrates, 50 messages of 20 bits in each case, and measure how the chosen bitrate affects the resulting bit error rate (BER). We considered BER of 10% as suitable, since in this case error-correcting code can be applied to ensure correct decoding with manageable overhead. Sounds are produced on the Apple HomePod speaker.

The results for five bitrates are shown in Table 17. The payloads can be successfully transmitted at bitrates up to 5bit/s. At a higher bitrate, the number of samples within a

Table 17: Transmission BER for selected frequencies & SPLs.

Frequency	SPL, dB (at 1kHz)	BER, % 1bit/s	2bit/s	5bit/s	10bit/s
18Hz	72	0.5	4.8	5.0	20.0
18Hz	65	4.8	8.0	17.5	17.8
20Hz	65	1.8	3.2	16.0	27.0

recording time frame for each bit becomes too small to correctly identify the frequency of vibrations. As a result, we consider bitrates of 2–5bit/s as practical.

6.5 COUNTERMEASURES AND DISCUSSION

COUNTERMEASURES. The most straightforward way to prevent the proposed covert channel is to apply a high-pass filter to all sounds played by the speaker, on the hardware or OS level, cutting off all frequencies below 24 Hz. In that case, transmission would be still possible using higher frequencies, but may not be inaudible. However, the attacker may still try to apply a steganographic approach, hiding hardly-perceivable signals into seemingly benign sounds, e.g., music. Alternatively, effort can be put on reducing vibrations conducted from the speaker's enclosure to the surface, e.g., by using anti-vibration pads. However, it may be not possible to prevent the vibrations completely, as the working principle of the speaker woofer is based on physical vibrations of the diaphragm.

The high-pass filter can also be applied to the accelerometer measurements to prevent detection of the low-frequency signals in the spectrum domain, although precise sensor measurements can be required for some apps. Furthermore, mobile OS vendors can forbid access to accelerometer data without an explicit permission granted by the user, limiting the applicability of the covert channel.

FUTURE WORK. First, we would like to point out that the transmission requires the transmitter and receiver to share a common surface and does not work over the air, at least at tested SPLs. We tried placing the smartphone very close to the speaker (≈5cm) on another surface, and did not observe the signal. Furthermore, the signal depends on the surface itself: We confirmed that decoding worked for 2 different office tables and 5 different relative positions of devices on the same table, with 10–160cm between devices. However, the transmission did not work on more stable surfaces (e.g., a window sill from artificial stone) or softer surfaces (e.g., a moquette floor). More extensive evaluation of the depending of the signal on the surface may be needed to confirm applicability of the approach. Second, the low transmission rates can be improved, by investigating whether more efficient encoding schemes can be applied (e.g., multiple frequency shift keying), and more accurate signal synchronization can be implemented. Finally, we did not thoroughly investigate the nature of vibrations produced by the speakers. Comparing speaker designs in terms of produced vibrations can be direction for future work.

6.6 SUMMARY

In this chapter, we presented a vibrational covert channel between devices equipped with speakers and mobile devices. We showed that binary data can be encoded into inaudible low-frequency sounds. When playing these sounds, speakers produce slight vibrations of the surface, which can be captured by the accelerometer on the mobile device. The presented covert channel does not require any explicit permissions and can be used unnoticed to end users. We identified suitable ranges of frequencies for such transmission, and evaluated the basic transmission rate.

7

TOWARDS SYSTEMATIC IDENTIFICATION OF CROSS-COMPONENT COVERT CHANNELS

In this chapter, we propose an approach to automatically identify and evaluate covert channels that are based on disruptive physical interference between hardware components of mobile devices. For this purpose, we design an evaluation framework that triggers activities of specific hardware components of an Android mobile device, while simultaneously collecting activity traces of other components (e.g., measurements of motion sensors). Afterwards, we apply anomaly detection methods to identify if collected traces are disrupted by the triggered activities, and determine if the observed effect can be exploited to construct a reliable covert channel. Furthermore, we demonstrate how the same approach can be used to estimate potential capacity of detected covert channels. The proposed solution makes it possible to detect and compare different cross-component covert channels in a unified manner, independently of the nature of the physical influence between components. As a result of the experimental evaluation on 20 smartphones for selected hardware components, we are able to identify 3 previously known sensor-based covert channels (including the channels introduced in Chapters 2–5), confirm 5 other sensor-based attacks to be applicable for covert channels on mobile devices, and identify 4 new covert channels.

Remarks. Kristian Kullmann has implemented parts of the backend for the evaluation framework and ported the framework to the iOS platform in his bachelor's thesis [Kul19].

7.1 MOTIVATION AND CONTRIBUTIONS

In the previous chapters, we demonstrate several covert channels in mobile devices based on the reaction of motion sensors to disruptive activity of other electronic components of the device: We utilized the magnetometer's reaction to the electromagnetic (EM) emanations produced by the CPU (Chapters 2–3), and the reaction of the gyroscope and accelerometer to the device's loudspeaker producing disruptive high- and low-frequency sounds, respectively (Chapter 5). These examples also illustrate that physical interference of different nature can be exploited to construct covert channels applicable to similar application scenarios: For example, both EM- and sound-based covert channels can be used to establish web-tracking on a smartphone. As mobile devices continue to evolve and incorporate new hardware components and sensors, it will be necessary to proactively identify covert channels based on cross-component interference and compare their feasibility to different scenarios.

Furthermore, we also observe that interference that is detected on a particular device may not be detectable on another device model, depending on individual characteristics of the components, their mutual location within the device, and OS settings. For example, as we discuss in Chapter 5, gyroscope sensors may react to ultrasounds of different frequencies, and the speaker and digital-to-analog converter (DAC) may or may not be able to produce these frequencies without aliasing. Similarly, in Chapter 4 we show

that CPU EM activity on some devices does not affect their magnetometers. Therefore, many device models have to be evaluated to assess the overall feasibility of potential covert channels in a real-world setting. These examples further demonstrate the need for systematic and automatized approaches that can identify cross-component interference and evaluate if covert channels exploiting this interference can be constructed.

Considering the aforementioned factors, in this chapter we propose an approach that allows us to automatically detect disruptive covert channels on mobile devices. For this purpose, we generalize the cross-component influence between hardware components as ability of certain hardware components to cause measurable disruption in the traces of other, reacting components (e.g., mobile sensors). We implement a framework that triggers activities of potentially affecting hardware components of the device and monitors sensor traces in a synchronized and controllable manner. Afterwards, we apply the outlier detection methods to detect if sensor traces are affected by the activity of the affecting components, and to estimate the capacity of the covert channel utilizing this effect.

The implemented framework has a client-server modular architecture (detailed in Section 7.2) that allows us to test multiple devices simultaneously. Furthermore, it is possible to run the evaluation for devices running in the cloud device farms, such as Visual Studio App Center [Vis20]. Although in this chapter we only focus on detecting intra-device cross-component interference, the framework can also be applied to detect and evaluate cross-component influence between components of two different devices, or be extended to test reaction to other external interference sources (not limited to activity of hardware components of mobile devices). We leave extension of the framework to inter-device covert channels for future work.

In the experimental evaluation of our method, we test 17 pairs of components on 20 different models of modern smartphones. The validity of our method is confirmed by successfully detecting 3 previously known cross-component influences (including the covert channels presented in Chapters 2–5). Furthermore, we have identified 5 covert channels that utilize interference sources examined in related works for other sensor-based attacks, or for covert channels in different scenarios (e.g., for inter-device covert channels, or covert channels not utilizing smartphones). Finally, we have observed 4 novel cross-component interference sources that can be utilized to establish covert channels, that, to the best of our knowledge, were not described in literature before. These findings are the following:

- On selected smartphones, the smartphone's central processing unit (CPU) can affect gyroscope and accelerometer sensors. Furthermore, the device's screen can be used as an alternative source of EM interference affecting device's sensors.
- A barometer sensor on selected devices can be affected by the speaker producing inaudible sounds.
- On one tested device, we observe the reaction of the microphone to CPU activity.

Overall, the evaluation results confirm the applicability and validity of our model to detect and evaluate cross-component covert channels on mobile devices.

Table 18: Summary of cross-component interference sources discovered or confirmed by our framework as a result of the experimental evaluation. Our contribution is compared to existing related works.

Affecting component	Reacting component	Our contribution[a]	Related works[b]	
CPU	Magnetometer	Confirmation	CCO	Chapter 2
			CCM	Chapter 3
			SC	Chapter 4
			CCO	Guri [Gur21b]
			SC	Cheng et al. [Che+19]
CPU	Accelerometer	New source		
CPU	Gyroscope	New source		
CPU	Microphone	Covert channel	SI	Kune et al. [Kun+13]
Screen	Magnetometer	Covert channel	SC	Ning et al. [Nin+18]
Screen	Accelerometer	New source		
Speaker	Magnetometer	Covert channel	SI	Nashimoto et al. [Nas+18]
Speaker	Gyroscope	Confirmation	CCO	Chapter 5
			CCO	Farshteindiker et al. [Far+16]
			SI	Wang et al. [Wan+17]
			SI	Nashimoto et al. [Nas+18]
Speaker	Accelerometer	Confirmation	CCO,SI	Trippel et al. [Tri+17]
			CCM	Block et al. [BNN17]
Speaker	Barometer	New source		
Speaker	Camera	Covert channel	SI	Ji et al. [Ji+21]

[a] **New source** — we discover the interference source not observed in related works, and apply it to establish covert channels.
Covert channel — we detect the interference used in other scenarios or domains, and apply it to establish covert channels.
Confirmation — we detect and evaluate the covert channels presented in related works.
[b] **CCM** — cross-component covert channel on mobile devices; **CCO** — covert channel in other scenarios or domains; **SC** — side-channel attack; **SI** — signal-injection attack.

7.1.1 *Related work*

Several interference sources identified by the proposed framework have been previously explored in related works. These works include covert channels in different scenarios, but also passive side-channel attacks exploiting interference as the source of information leakage, and signal-injection attacks targeting the integrity of the sensor output. A summary of newly identified covert channel sources, as well as the sources previously described in related works on covert channels, side-channel and signal-injection attacks, and identified by the framework, is presented in Table 18.

Similarly to our findings in Chapter 2, Guri et al. [Gur21b] proposed to use magnetometers in smartphones as receivers for covert signals from nearby computers, where

the signal is encoded into EM emanations caused by the CPU. Cheng et al. [Che+19] proposed to utilize this source to establish a side-channel attack that identifies applications or webpages running on a target laptop in vicinity of a smartphone. As we further demonstrate in Chapters 3–4, this interference can be used to establish *intra-device* covert- and side channels on selected devices. Ning et al. [Nin+18] showed that activity of an LED display of a smartphone can affect its magnetometer, and utilized this reaction as a side channel to infer running applications. Our evaluation work confirms this reaction on multiple devices. In addition, we observe that accelerometers can be also affected by the device's CPU and screen's activity on selected devices. Kune et al. [Kun+13] demonstrated that microphones in consumer devices can be susceptible to injection of specially crafted EM signals. Our framework identifies one device on which microphone can be affected by its CPU activity, presumably reacting to EM emanations from the CPU.

Several works have studied how susceptibility of MEMS gyroscopes and accelerometers to acoustic interference can be exploited to intentionally disrupt sensor measurements, or to construct a covert channels. In particular, Block et al. [BNN17] proposed an intra-device covert channel utilizing the reaction of smartphone's accelerometers to ultrasonic resonant sounds played by the smartphone's speaker. Farshteindiker et al. [Far+16] presented an inter-device covert channel scenario with a special ultrasonic transducer as the transmitter and a gyroscope as the receiver. Further works are listed in Chapter 5. Our evaluation using the proposed framework confirms applicability of the speaker-to-gyroscope covert channels to different smartphone models. Furthermore, on selected devices, we identified that smartphone's accelerometer and barometer sensors can also react to ultrasonic sounds played by the speaker.

Multiple research works exploited acoustic susceptibility of MEMS gyroscopes and accelerometers for signal-injection attacks in different scenarios [Nas+18; Tri+17; Wan+17]. Nashimoto et al. [Nas+18] additionally observed reaction of magnetometers to acoustic interference, that we also confirmed on selected smartphones. Finally, in the recent work, Ji et al. [Ji+21] studied the acoustic attack on image stabilization of the cameras, targeting the computer vision systems in the autonomous vehicles. In our evaluation, we also identified the reaction of cameras to ultrasonic sounds on some devices, caused by the disturbance in the gyroscope sensors used for image stabilization.

7.2 METHODOLOGY

In this section, we describe our methodology. We start by introducing the model of cross-component covert channels that we use, and outline the detection steps. Afterwards, we describe each step of the system: the architecture of the framework for collecting measurements in a controllable manner, the details of the feature extraction and selection, the applied novelty detection method, and our approach to estimate the channel capacity.

7.2.1 *Model of cross-component interference*

In this work, we consider a generic scenario for intra-device covert channels with two user-level processes, a transmitter and a receiver. In order to generalize cross-component covert channels, we further divide hardware components of mobile devices into two sets: *affecting components* $\{c_a\}$ and *reacting components* $\{c_r\}$. We assume the transmitter can

trigger activity of the affecting components. The receiver can collect activity traces of the reacting components as one or several time series, i.e., discrete numerical observations ordered in time. For example, the digital recording of the microphone's output after applying the analog-to-digital converter (ADC) represents its activity trace; for motion sensors, their 3-axis outputs represent their activity traces.

We identify that cross-component interference between two components $\{c_a, c_r\}$ exists if the triggered activity of the affecting component c_a introduces observable (detectable) interference in the corresponding traces of the component c_r within a certain time interval. If such cross-component interference exists, a covert channel between a transmitter and a receiver can (at least) be constructed in an On-Off manner: within each time frame of the specified length t, the component c_a can trigger affecting activity to encode a 1, or perform no activity to encode a 0. Subsequently, the receiver tries to detect interference within each time frame to decode the signal bit by bit.

Our main idea is to detect the cross-component interference by extracting features from the recorded traces and applying the semi-supervised anomaly detection methods, also referred to as novelty detection. The novelty detection classifier will assign an outlier score to each sample to be classified, based on the trained model of the "normal" data. In presence of the interference, the affected traces will be classified as anomalous (i.e., their outlier scores exceed a threshold). Using this model, the receiver can apply the novelty detection to each time frame of the received signal. If the trace within the time frame t is classified as the outlier, the receiver decodes it as a 1. If the trace is not classified as an outlier, the receiver decodes it as a 0. In this covert channel, the transmission error rate will be determined by a number of normal traces that were incorrectly classified as outliers, and vice versa. Therefore, the primary goal is to determine the optimal threshold value for the outlier score the receiver can use to decide how to decode a particular trace. If an outlier score for a particular trace is below the threshold, transmitter considers it to be not affected and decodes a bit 0. If the outlier score is above the threshold, the transmitter considers the interference to exist and decodes a bit 1.

Note that this model generalizes all covert channels presented in the previous chapters of this dissertation. For example, in the magnetic covert channel described in Chapter 3 the CPU is utilized as an affecting component, and a magnetometer as a reacting component. Change of the amplitude in recorded traces can be used as the distinct feature in the novelty detection algorithm to detect interference and decode the signal. Similarly, for the acoustic covert channels described in Chapter 5, the speaker is used as an affecting component producing ultrasonic resonance sounds, while the gyroscope is used as a reacting component. The variance of the gyroscope measurements along a particular axis, or the magnitude of their resulting FFT-coefficients (as described in Section 6.3) can be used as features in the novelty detection to detect interference and to decode the signal.

For some pair of components, not every activity of the affecting component generates the interference to a certain affecting component. For example, for the speaker-to-gyroscope covert channel (Chapter 5), the speaker must produce the sounds in the specific frequency range to disturb the sensor measurements. Therefore, our approach identifies the interference only if the triggered activity of affecting components does affect the reacting component. In our experiments, we aimed to stress the affecting components to their peak activity (e.g., cause up to 100% of the CPU load), in order to produce the

highest possible interference. Section 7.3.1 provides details about the tested components and implementation of their activities in detail.

7.2.2 *Overview of detection methodology*

Given the introduced model that generalizes disruptive cross-component covert channels, the entire process of detecting a covert channel utilizing a pair of components $\{c_a, c_r\}$ consists of the following steps, that we describe in detail in the following sections:

1 **Data collection** (Section 7.2.3). For each pair of the affecting and reacting components, the framework collects the activity traces of the reacting component, with and without simultaneous activity of the affecting component.

2 **Feature engineering** (Section 7.2.4). We extract a vector of temporal, spectral and statistical features from the recorded traces.

3 **Novelty detection** (Section 7.2.5). We train the novelty detection model that learns the state of normality for the traces recorded without the affecting activity. Afterwards, we apply the model to classify the traces with and without the affecting activity as anomalous or not, and compute their outlier scores.

4 **Covert channel detection** (Section 7.2.6). We analyze distributions of the outlier scores for non-affected and affected traces to determine a threshold outlier score value, that can be used to decode the on-off-encoded signal bit by bit.

5 **Capacity estimation** (Section 7.2.7). We estimate the capacity of the covert channel by varying the interval of the recorded traces and repeating the novelty detection and covert channel detection steps.

7.2.3 *Data collection*

DATA COLLECTION FRAMEWORK. To collect traces of the reacting components alone, or in presence of the activities from the affecting components, we develop a data collection framework. The framework has a client-server architecture, and comprises an Android mobile application that controls affecting and reacting components on target devices, a remote control script instrumenting the experiments, and an HTTP server responsible for communication between the mobile application and the control script and for receiving and storing the resulting traces for further offline analysis.

Each individual recording experiment is started by the control script. In particular, the task request specifies the affecting component(s) to be triggered, the reacting component(s) to be traced, duration of affecting and reacting activities, and their custom parameters. One can also specify individual delays for affecting and reacting activities. It allows us to perform initialization of both components before starting their activities (e.g., initialization of the camera can take up to a few seconds). Custom delays also allow us to start tracing the reacting component before the affecting activity starts, in order to ensure that start of the affecting activity is fully captured in the recorded trace.

The main component of the framework is an Android mobile application. The application registers a device with the server and waits for new task requests. After receiving

the task request, the application pre-initializes the components, triggers the components in a synchronized manner (taking into account individual delays and durations), and finally sends the recorded traces back to the server. The framework has an extendible modular architecture, i.e., new affecting and reacting components can be added as subclasses of the base component class of the Android application. Each new module only declares the function that is responsible for triggering or tracing the component activity, respectfully, and for the component initialization. We aim to minimize CPU activity and other execution side effects when performing each experiment. This is an important step since reacting components can be affected by the CPU. In particular, all components are to be pre-initialized before starting the experiments, and no processing or network communication to be performed until the components finish their execution.

A client-server architecture allows us to run experiments on multiple mobile devices at the same time. In this case, a control script issues the same task request that is to be executed by every device requesting the task from the server. In particular, to perform experiments on many smartphones, we make use of cloud platforms that allow us to run mobile applications on multiple mobile devices remotely. For our experiments, we use the Microsoft Visual Studio App Center [Vis20] platform.

Finally, although we focus on intra-device cross-component interference in this chapter, the framework can also evaluate inter-device covert-channel interference as well, by sending a task to trigger affecting components on one device (transmitter), and a task to monitor the reacting component on another device. Furthermore, the framework can be extended to evaluate external interference sources, i.e., to implement external affecting components in addition to the activity of internal hardware components of the device. To achieve it, a computer controlling the external component must implement the client that will communicate with the server. In addition, in order to synchronize inter-device experiments, an exact Unix timestamp can be specified in the experiment to schedule the reacting and affecting activities at exact absolute time. To precisely control inter-device experiments, the application periodically synchronizes its clock with the public Network Time Protocol (NTP) server using the TrueTime library [Tru22]. This step is important as the system clock on mobile devices gradually drifts over time [GLB15].

COLLECTION OF ACTIVITY TRACES. For each pair of affecting and reacting components $\{c_a, c_r\}$, the framework collects three datasets each containing N traces of reacting component's activity: $\mathcal{D}_{norm}$, $\mathcal{D}_{test}$, and $\mathcal{D}_{anom}$. Datasets $\mathcal{D}_{norm}$ and $\mathcal{D}_{test}$ contain traces of c_r that were recorded without activity of c_a, while the dataset $\mathcal{D}_{anom}$ contains traces that were recorded simultaneously with the activity of c_a. The dataset $\mathcal{D}_{norm}$ will be used to train the model of normality by the novelty detection algorithm, while $\mathcal{D}_{test}$ and $\mathcal{D}_{anom}$ are used to analyze if the affecting component affects the traces in $\mathcal{D}_{anom}$, and to analyze distributions of their outlier scores for determining existence of the covert channel.

Note that we do not collect these datasets one by one. Instead, for each individual trace, we randomly choose whether this trace will be recorded with the affecting activity or not, until we fill in all three datasets. This approach allows us to eliminate the risks that certain background activity, external noise, or another component, would affect traces in the same dataset (e.g., all the traces in $\mathcal{D}_{anom}$), potentially leading to wrong classification.

Table 19: List of selected features, as defined and implemented in [Bar+20].

	Statistical features
F00	Histogram
F01	Kurtosis
F02	Max
F03	Mean
F04	Mean abs. deviation
F05	Median
F06	Median abs. deviation
F07	Min
F08	Root mean square
F09	Skewness
F10	Standard deviation
F11	Variance

	Spectral and temporal features
F12	Absolute energy
F13	Area under the curve
F14	Autocorrelation
F15	Centroid
F16	Mean absolute diff
F17	Total energy
F18	Zero crossing rate
F19	FFT Magnitude
F20	Max power spectrum
F21	Median frequency
F22	Spectral centroid
F23	Spectral kurtosis
F24	Spectral skewness

7.2.4 *Feature engineering*

For each trace in the three datasets $\mathcal{D}_{\mathtt{norm}}$, $\mathcal{D}_{\mathtt{test}}$, and $\mathcal{D}_{\mathtt{anom}}$, we extract the vector of features characterizing the trace. For this purpose, we utilize the open-source framework *tsfel* [Bar+20], designed to effectively extract features from the time series data. We select a subset of features implemented by the framework that includes temporal, spectral, and statistical features. Most of the features return a single scalar value, while some features output a vector of values (e.g., a histogram counts the number of samples within predefined number of bins). In addition, we implement a custom feature *F19* that computes the average magnitudes of the signal's positive-frequency terms of the Fast Fourier Transform (FFT) within predefined number of bins, to precisely capture differences in the signal's spectrum. A complete list of features we used in our experiments is shown in Table 19. If a recorded trace contains multiple time series (e.g., the output of 3-axis measurements of motion sensors producing three time series for each axis), we extract the features for each time series and merge the results. As a result, the total number of features is equal to the number of features for one time series multiplied by the number of time series. After the features are extracted from time series, we evolve additional step to reduce the dimensionality of the feature space, by excluding features that have a very small variance for all recorded traces. In our experiments, we set a threshold of 10^{-12} for feature's minimum variance. In future work, we plan to test other methods for dimensionality reduction.

An additional preprocessing step is required for video recordings, to convert recorded video stream to multiple time series. A straightforward approach to convert a video stream to a set of time series would be to consider the content of every pixel of the video frame as individual time series, with a time vector calculated based on the video frames per second (FPS). However, such approach results in a very large number of time series and the resulting features to be extracted (e.g., there are 2073600 pixels in a video of Full HD resolution), that exceeds practical computational complexity. Therefore, for

video recordings, we first perform a pre-processing step that extracts from the original video several time series that characterize video frames. In our experiments, we compute variances of Sobel and Laplacian filters. These methods are typically used to estimate the amount of motion and level of details and noise in the video stream [PPG13; Ros15].

As a result of the feature engineering with M features in total, we describe each trace as a vector in M-dimensional feature space, and construct three feature matrices F_{norm}, F_{test}, and F_{anom} of $M \times N$ dimension, that contain features vectors for each trace from the datasets $\mathcal{D}_{norm}$, $\mathcal{D}_{test}$, and $\mathcal{D}_{anom}$, respectively.

7.2.5 *Novelty detection*

To identify if traces in $\mathcal{D}_{anom}$ are affected by the activity of the affecting component, we apply the novelty detection methods, also referred to in literature as *semi-supervised anomaly detection* or *one-class classification*. It is the subset of machine learning algorithms which construct a model of "normal" data based on the training samples, also called the *model of normality*, and aim to classify whether new unseen samples differ from the samples in the training data, i.e., if new samples are *abnormal*, or *anomalous*, with respect to the model. Typically, novelty detection algorithm assigns an outlier score (sometimes also referred to as novelty score) to the test samples, that defines the "abnormality" of the samples. An overview of novelty detection algorithms can be found in [Pim+14]. Unlike supervised machine learning algorithms with two or more classes, the training phase in novelty detection setup contains only one class (normal), while abnormal samples are not known *a priori* and are not included in the training data. This complies with the goals of our approach, as we target to identify any possible interference without prior knowledge about the nature of this interference (assuming it can be different from known interference sources).

In our approach, we rely on the Local Outlier Factor (LOF) algorithm [Bre+00], implemented within the PyOD toolkit [ZNL19]. The algorithm assigns the LOF score to each point, that can be defined as the ratio between the average local density of the point's nearest neighbors to the local density of the point itself. Intuitively, if the point lies outside the local cluster of points, its local density is lower than of its nearest neighbors, and the LOF score is greater than 1. The formal definition of the algorithm can be found in [Bre+00]. A threshold value for the LOF serves as a decision boundary that classifies new test samples to be outliers or not based on their LOF scores with regard to their neighbors. In case of novelty detection, the LOF is calculated for each individual point with respect to the points from the training set representing normality. For points within the normality set, the LOF is also computed with respect to all other points in this set. In our case, the novelty detection algorithm builds the model of normality in the multidimensional feature space based on vectors from F_{norm}. Afterwards, it computes the LOF scores for both vectors from F_{test} and F_{anom}, denoted as LOF_{test} and LOF_{anom}, respectively. To verify that the algorithm generalizes the state of normality correctly (so that new non-affected samples have the LOF scores close to 1), we compare the mean LOF-score computed for F_{test} with a threshold. Finally, the resulting distributions of LOF_{anom} and LOF_{test} can be used to analyze the cross-component interference.

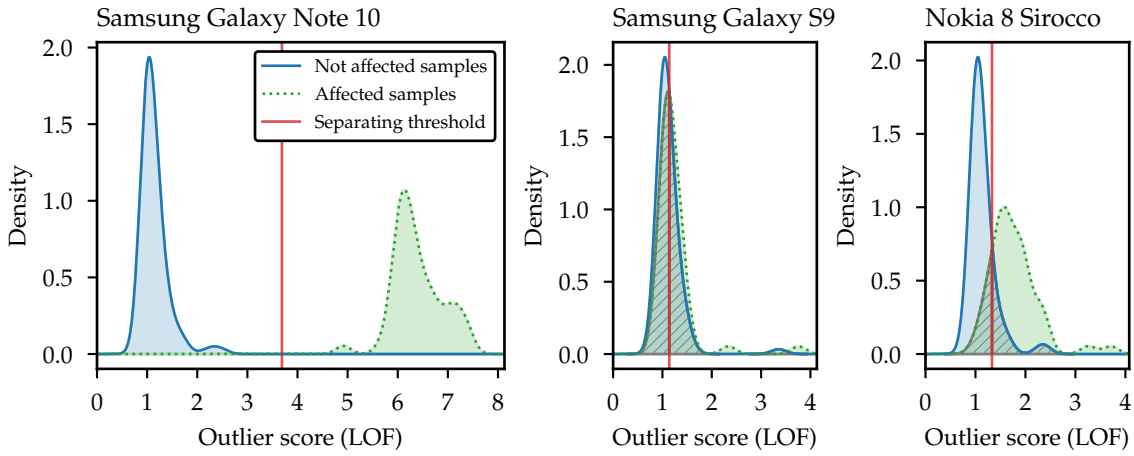


Figure 26: Example of the kernel density estimates (KDEs) of the outlier score distributions for non-affected (non-anomalous) and affected (potentially anomalous) samples. The samples represent magnetometer traces and the CPU affecting component on three smartphones. On device **(A)**, the covert channel is detected as distributions are completely separable. On device **(B)**, no interference is detected. On device **(C)**, the error coefficient is too large to utilize it for a reliable covert channel.

7.2.6 *Covert channel detection*

Although the statistical difference between the distributions of outlier scores LOF_{anom} and LOF_{test} can alone signify the existence of cross-component interference, it may not be possible to exploit this interference to establish a covert channel with a practical bit error rate. Instead, we propose a heuristic approach to estimate the optimal threshold value that minimizes the probability of a trace to be incorrectly decoded by the receiver. Essentially, it means minimizing the probabilities that a non-affected trace has an outlier score larger than the threshold (false positive) or an affected trace has an outlier score lower than then threshold (false negative).

In order to find this optimal threshold, we calculate the kernel density estimates (KDEs) of the probability density functions (PDFs) for both distributions of outlier scores LOF_{test} and LOF_{anom}. We use the KDE implementation of the scikit-learn framework [Ped+11]. The optimal kernel and its bandwidth (in our experiments, Epanechnikov kernel with the bandwidth of 0.15) are determined using the grid-search cross-validation for the training dataset (D_{norm}).

Given the KDEs for LOF_{test} and LOF_{anom}, the problem is reduced to the task of finding the outlier score that minimizes the sum of the area of KDE_{test} located to the right of the threshold value (probability of a non-affected trace to have an outlier score larger than a threshold), and the area of KDE_{anom} located to the left of the threshold (probability of an affected trace to have an outlier scores below the threshold). Formally, we aim to find the outlier score $x_{threshold}$ that minimizes the following parameter that we call the *estimated bit error rate (eBER)*:

$$eBER == \min_{x} \left(\int_{0}^{x} KDE_{anom}(t)\,dt + \int_{x}^{LOF_{max}} KDE_{test}(t)\,dt \right) \tag{1}$$

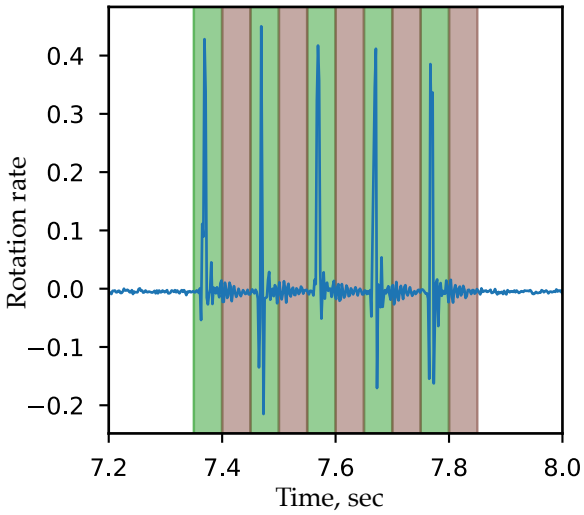


Figure 27: Example of the transmission rate estimation. Green areas represent the time intervals that include activity of the affecting component. Gray areas represent intervals without activity of the affecting component. Side effects can be observed in the beginning of the gray intervals.

where LOF_{max} is the maximum computed outlier score, and KDE_{test} and KDE_{anom} are KDEs for LOF_{test} and LOF_{anom}, respectively. To find $x_{threshold}$, we solve the global optimization problem by applying the Basin-hopping algorithm [Leaoo]. As a result, the $e\mathrm{BER}$ gives a heuristic estimate for the covert channel's bit error rate (BER). By setting a threshold for $e\mathrm{BER}$, we decide if a covert channel can be constructed based on the interference between the two components. In our experiments, we used a threshold of 0.1, assuming that error-correcting codes can be used to correct the estimated BER of 10%.

The Figure 26 shows examples of KDEs for traces of the magnetometer sensor and the affecting CPU activity, collected on three different smartphones. The Figure indicates non-affected samples (blue), affected samples (green), as well as the discovered $e\mathrm{BER}$ (dashed area) and the $x_{threshold}$ (red vertical line). On the left picture, the distributions are completely separable with $e\mathrm{BER} = 0$, and we conclude that a covert channel can be established based on this cross-component interference. On the middle picture, no cross-component interference is detected, as distributions of outlier scores are not separable and $e\mathrm{BER} > 0.1$. Finally, on the right picture, the distributions of outlier scores are clearly statistically different, signifying that cross-component interference exists between two components. However, $e\mathrm{BER} > 0.1$, therefore, we cannot construct a covert channel based on this interference.

7.2.7 *Capacity estimation*

The method described above allows us to detect and, if cross-component interference exists with $e\mathrm{BER} < 0.1$, to construct an OOK-based covert channel with a capacity of $1/t$, where t is the duration of each recorded trace. Therefore, the maximum capacity of covert channels is determined by the shortest time interval t_{min}, within which the affecting

component can reliably produce cross-component interference that can be captured by the anomaly detection algorithm. To empirically identify the maximum capacity of the covert channel, one can reduce the duration t and repeat the collection of traces and calculation of the estimated bit error rate $eBER$ until it exceeds a predefined threshold.

However, several peculiarities have to be taken into account. First, on the transmitter side, the affecting activity should not produce *side effects* within the recording trace *after* the time interval t is elapsed, as the next interval may correspond to the non-affected state and encode a 0 in the continuous transmission of the binary sequence. Generally speaking, the activity of the affecting component in the next interval can be correctly interpreted even in presence of side effects, as long as observed effects are reliably distinguishable from the intervals corresponding to the active triggering state. However, for simplicity, we set a stronger assumption and require no side effects, leaving more complex analysis for future work. Second, on the receiver side, the duration of t cannot be shorter than the sampling period of the receiving component. Finally, the collected trace should contain enough samples so that the interference will be detected by the algorithm.

We propose the following approach to empirically estimate practical capacity of the covert channel, that extends the detection methodology described above. For this purpose, we modify a different data collection procedure. Instead of triggering an affecting activity of c_a and monitoring the reacting activity for each trace, the affecting component c_a alternates its state between the triggered activity for a time t, and the idle activity for a time t. The activity trace of c_r is collected for this whole sequence. Figure 27 demonstrates one such recorded sequence with corresponding active and idle intervals. Afterwards, we apply the anomaly detection to every interval corresponding to the active state of the affecting component, and idle state of the affecting component, and verify that these intervals are correctly decoded to 1 and 0, respectively (allowing a small percentage of errors). This way, we ensure that the intervals corresponding to the active state are still classified as outliers, while ensuring that intervals corresponding to the idle activity correspond to the state of normality, and side effects do not influence the transmission. We repeat the same experiment while gradually reducing the duration of t, until the detection algorithm does not correctly classify the intervals.

Two parameters have to be additionally considered in our approach. First, the transmitter can control the duration of the affecting activity t_a within the target interval t, where $t_a \leqslant t$. Potentially, an activity lasting shorter than t can generate interference within the entire interval of t due to remaining side effects after the activity is triggered. Therefore, several parameters for t_a can be tested when evaluating the capacity, selecting the best results (e.g., in our experiments, we test t_a equal to t, $0.8t$, $0.6t$ and $0.4t$). Second, as we observe in our experiments, reaction of the reacting components to the affecting component can be *delayed* compared to the beginning of the affecting activity (e.g., we observe that the acoustic noise causes the resonance interference to the vibrating mass of the gyroscope with a slight delay). Therefore, it is important to consider potential delay when evaluation the capacity. In our experiments, we further run the analysis of the recorded traces considering small predefined delays as fractions of the target interval t (e.g., $0.1t$, $0.25t$ and $0.5t$).

7.3 EXPERIMENTAL EVALUATION

In this section, we present the evaluation results of our method. First, we describe selected affecting and reacting components to be tested, and present results of cross-component interferences identified for these components on 20 tested devices. Afterwards, for selected pairs of affected components, we estimate capacity of the covert channels.

7.3.1 *Tested hardware components*

In this section, we provide some technical details about the selected affecting and reacting components. As affecting components, we have selected the device's CPU, screen, and the speaker. As reacting components, we have selected magnetometer, gyroscope, accelerometer, and barometer sensors, as well as device's microphone and camera. Below we describe some implementation details about these components.

CPU. To trigger peak CPU affecting activity, we follow an approach proposed in Section 3.3, and produce *busy waiting* loops in a number of threads equal to the amount of available logical processor cores, utilizing this way up to 100% of the CPU time. As the CPU activity affects some of reacting components, we implement all other affecting components in such a way that their activity aims to minimize indirect CPU activity after the component is initialized.

Screen. We consider the smartphone's screen as another potential source of the EM emanations. We implement two types of the affecting screen activity. The first type adjusts the brightness of the screen from the minimum value (set at the initialization phase) to the maximum value. The second type instead alternates the content of the screen: at the initialization phase, the component shows an image containing pixels of one color (e.g., a full-screen black screen); When triggered, the component shows another image containing pixels of another color (e.g., a white screen). This approach is motivated by the fact that modern mobile OLED screens consume different amounts of power depending on the color of pixels [Che+13]. The images and corresponding UI components are preloaded at the initialization phase to avoid unnecessary CPU noise when the activity starts.

Speaker. For the device's speaker, we generate a sound file containing a sine sweep of the predefined frequency range in ultrasonic range, i.e. a continuous sine wave with the frequency increasing linearly with time. We consider the sweep in the ultrasonic range (18.5kHz–24kHz), as it allows the transmitter to trigger activity inaudibly, without user's attention. Note that we also do not cover frequency ranges above 24kHz as most of mobile devices do not support high-resolution audio and have a sampling rate limited to 48kHz.

Additionally, we apply a Hanning window [Opp99] at the beginning and the end of the generated signal, to prevent audible clicks (see Section 5.4). The speaker component then plays the sound file at the system volume level of 80%, to avoid potential distortions and noises that we observe on some devices at higher volume levels.

Reacting components. As reacting components, we select magnetometer, gyroscope, accelerometer, and barometer sensors. These sensors are accessed using the Sensor API [Sen18]. If available, uncalibrated versions of motion sensors are used. In addition, we also include to the list of selected components a microphone, that records sounds at

Table 20: Cross-component interference for the CPU, Screen, and Speaker affecting components and different sensors as reacting components: magnetometer (mag), gyroscope (gyr), accelerometer (acc), barometer (bar), microphone (mic), and camera (cam). Devices `D01-D03` are lab devices. Dash signifies no influence detected. Camera can be evaluated only on lab devices. On all devices, barometer and camera were only affected by the Speaker; a microphone was not affected by the Screen; natural microphone reaction to the Speaker activity is omitted.

		CPU				Screen			Speaker				
	Device	mag	gyr	acc	mic	mag	gyr	acc	mag	gyr	acc	bar	cam
D00	Google Pixel 2	●	-	-	-	-	-	-	-	-	-	-	-
D01	Google Pixel 3	●	-	-	-	●	-	●	●	-	-	-	-
D02	Samsung Galaxy S8	-	-	-	-	-	-	-	-	●	-	-	●
D03	Samsung Galaxy S9	-	-	-	●	-	-	-	-	●	-	●	●
D04	Google Pixel 4	●	-	-	-	●	-	-	-	●	-	-	
D05	Google Pixel 4a	●	-	-	-	●	-	-	●	●	-	●	
D06	Huawei P30 Pro	-	-	-	-	●	-	-	-	●	-		
D07	LG G8S ThinQ	-	-	-	-	-	-	-	-	-	-	-	
D08	Motorola One Action	●	●	●	-	●	-	-	●	●	-		
D09	Motorola moto g(8) plus	-	●	●	-	●	-	●	●	●	●		
D10	Nokia 8 Sirocco	-	-	●	-	-	-	●	-	-	-	●	
D11	Nokia 8.1	●	●	-	-	●	-	-	-	-	-		
D12	OnePlus 8 Pro	●	-	-	-	●	-	-	-	●	-		
D13	OnePlus Nord	●	-	-	-	●	-	-	-	-	-		
D14	Samsung Galaxy Note10	●	●	-	-	-	-	-	-	●	●	●	
D15	Samsung Galaxy S20	-	-	-	-	-	●	-	-	●	●	-	
D16	Samsung Galaxy Tab S7+	●	-	-	-	●	●	-	●	●	●		
D17	Sony Xperia 5 II	-	●	-	●	-	-	-	-	●	●	-	
D18	Sony Xperia XZ2 Compact	-	●	●	-	-	-	-	-	●	-	-	
D19	Xiaomi Mi A3	●	-	-	-	-	-	-	●	-	-		

the sampling rate of 48kHz. Finally, we include the back camera, that records full-HD videos.

7.3.2 *Cross-component interference*

To evaluate a large number of devices, we conducted measurements using the cloud platform Visual Studio App Center [Vis20]. We selected 16 modern devices running Android 10 or Android 11. Additionally, 4 devices were used in the lab in a typical office environment. We could not control the environment of the devices in the cloud (such as their physical position on the surface, as well as ambient noise), and tested them as is. In particular, for this reason, we could not perform the experiments with the camera reacting component for devices in the cloud, as their camera is facing the surface they are lying on.

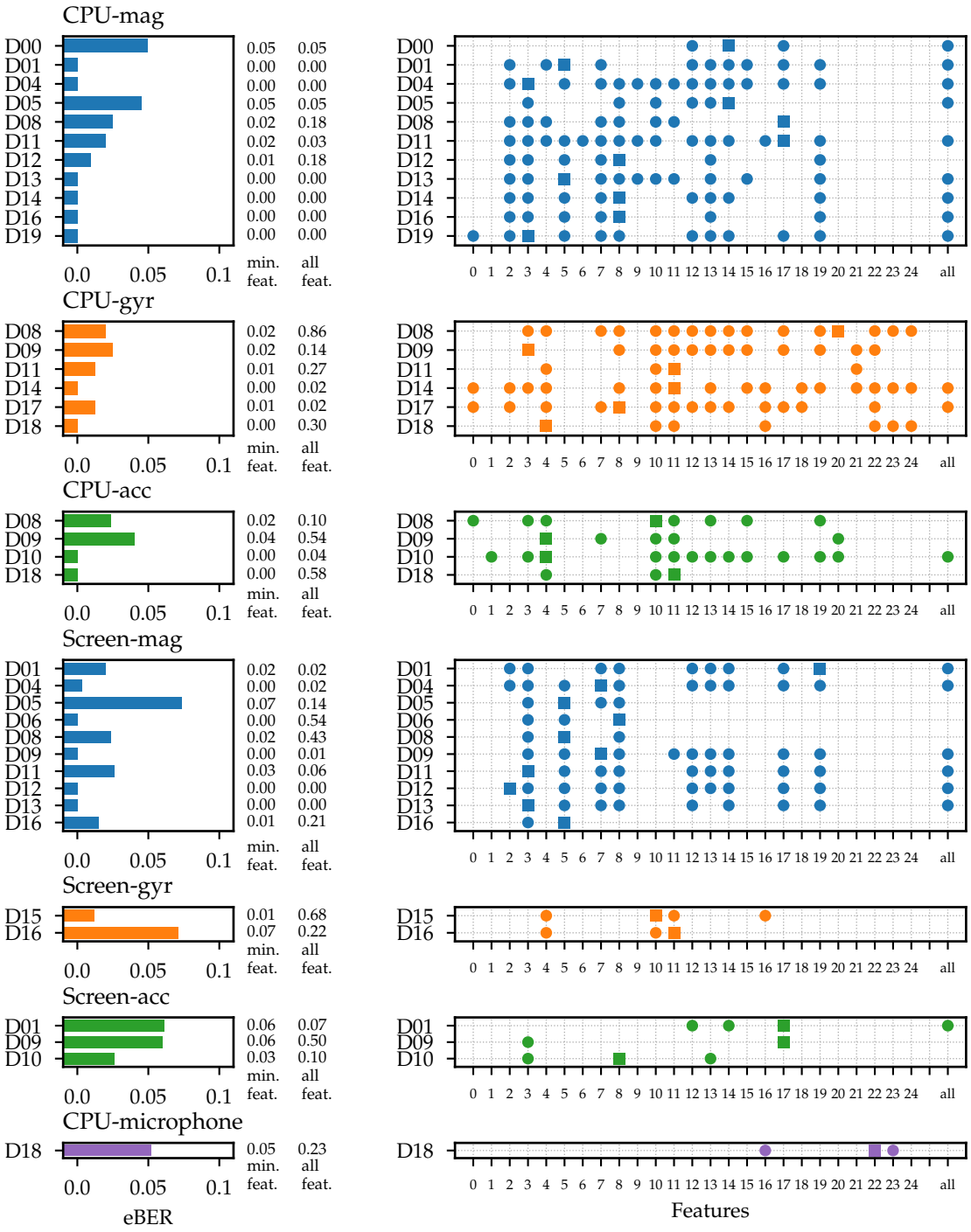


Figure 28: Estimated bit error rates (eBER) for pairs of components with the CPU and Screen as the affecting components. The figure compares the detection when using each individual feature type (Foo–F24), or all features combined (right). On the left, the minimal observed eBER is shown (for individual features), as well as eBER computed when all features are combined. The square point indicates the feature that results in the minimal eBER.

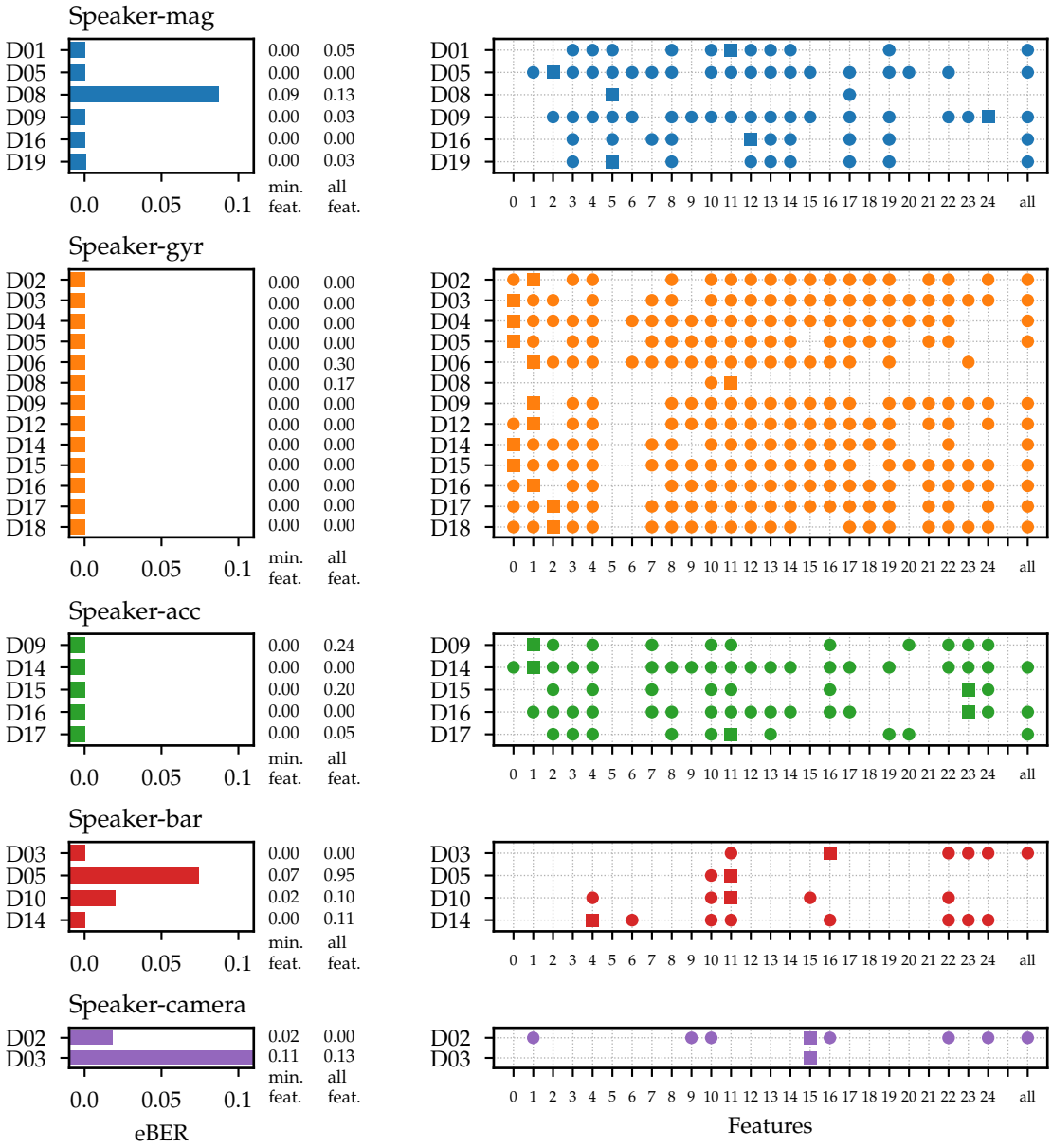


Figure 29: Estimated bit error rates (eBER) for pairs of components with the speaker as the affecting component. The figure compares the detection when using each individual feature type (F00–F24), or all features combined (right). On the left, the minimal observed eBER is shown (for individual feature), as well as eBER computed when all features are combined. The square point indicates the feature that results in the minimal eBER.

For each pair of tested components, we have collected traces with a duration of $t = 1$ second. We collected traces of three classes representing the normality state (D_{norm}), affected (D_{anom}) test traces, and not-affected test traces (D_{test}), with 50 traces per class. To evaluate how individual features influence to the detection and how well the novelty

detection performs with multidimensional feature space, we run the evaluation separately for each feature type, as well as for all features together.

Table 20 demonstrates the results for the cross-component covert channels on all tested devices for all pairs of components, where the covert channel was detected with $eBER < 0.1$ for at least one individual feature type, or when using all features combined. We can see that both electromagnetic and acoustic interference can influence different reacting components, and this reaction can be used to establish cross-component covert channels. The results confirm the findings in Chapter 4 and show that the CPU affects the magnetometer on more than half of tested devices. Our results demonstrate that device screen can be also used as alternative to CPU sources of electromagnetic interference, as they also affect the magnetometer sensor. Second, the accelerometer and gyroscope are also susceptible to electromagnetic interference from the CPU and screen on some devices. On many tested devices, gyroscopes and accelerometers are susceptible to acoustic interference. In particular, it confirms that intra-device scenarios for the covert channels introduced in Chapter 5 are feasible on a large number of devices. In addition, we have found that on selected devices barometer sensor can also be affected by ultrasonic interference. Finally, on two out of four tested devices, we have detected an acoustic-based covert channel with a camera as a receiver. Our preliminary analysis shows that the disturbance of the video stream is most likely caused by the acoustic susceptibility of the gyroscope that is used for the image stabilization of the video, as also observed in [Ji+21].

Figures 28 and 29 present a more detailed evaluation of each cross-component covert channel. In particular, we run the novelty algorithm for each feature type individually, as well as for all feature types combined. As one can see, for some pairs of components and for some devices, cross-component covert channels are detected only when applying the novelty detection with individual feature types (e.g., reaction of the gyroscope to the screen activity, or reaction of the microphone to the CPU). It indicates that the LOF algorithm used in our approach introduces false negatives in high-dimensional feature space. Nevertheless, considering each feature individually makes it possible to detect several additional cross-component covert channels.

Overall, we conclude that the approach we introduced in Section 7.2 allows us to successfully detect previously known, as well as new cross-component covert channels on many devices in a unified manner, that proves practical relevance and applicability of our method.

7.3.3 *Evaluation of the channel capacity*

In this experiment, we evaluate our approach to estimate the channel capacity, introduced in Section 7.2.7. For this purpose, we selected four different cross-component covert channels detected on four devices. For each covert channel, we first run our method to estimate the channel's capacity. We test bitrates with a step of 5bit/s (5bit/s, 10bit/s, 15bit/s, etc.). Afterwards, we implement the actual transmission of pseudorandom binary messages for different bitrates. We transmit 20 messages of 20bit length, compute the resulting bit error rate, and identify the maximum bitrate that results in the bit error rate (BER) below 10%.

The results are presented in Table 22. As one can see, the actual achieved capacity is close to the bitrate heuristically estimated by applying novelty detection. This confirms

Table 22: Estimated transmission rates for selected cross-component covert channels, as well as confirmed bitrates (with eBER and BER of less than 10%). Estimated bitrates are tested for 0.5, 1, 5, 10, 15, and 20bit/s. Actual bitrates are tested for 0.5, 1, 2, 5, 8, 10, and 12bit/s.

Components		Device	Estimated Bitrate	Confirmed Bitrate
Affecting	Reacting		(bit/s)	(bit/s)
CPU	magnetometer	D00	5	8
Screen	magnetometer	D01	1	1
Speaker	gyroscope	D03	10	12
Speaker	camera	D02	5	5

the applicability of our approach to estimate bitrate of cross-component covert channels, and to correctly identify the optimal threshold value to be used in the actual decoding of the signal. We leave the thorough evaluation of channel capacity and proof-of-concept transmission for remaining covert channels and for different devices for future work.

7.4 DISCUSSION

The approach we introduced in this Chapter serves as a foundation to automatic detection and evaluation of sensor-based covert channels in different settings. In this section, we outline possible extensions to this work.

First, the evaluation can be extended to include additional hardware components on Android devices; Furthermore, the same set of experiments can be also performed for iOS devices. As further candidates for affecting components, one can evaluate smartphone's cellular modem, Wi-Fi, Bluetooth, and GPS modules. We have conducted preliminary experiments showing that distinct isolated GPU activity (rendering complex graphics and performing general-purpose computations) can affect device's magnetometer on tested lab devices (Pixel 2 and Pixel 3). In addition, we have preliminarily confirmed that generation of cryptographic material using the Trusted Platform Module (TPM) on a Pixel 3 smartphone also affects device's magnetometer, although the reaction could be caused by the collateral activity of the main CPU. Furthermore, the evaluation setup can be extended to evaluate the inter-device physical interference, as well as reaction of sensors to other external sources, beyond activity of mobile components. The architecture of the developed framework already supports testing inter-device interference, when the affecting activity is triggered on one device, while the reacting activity is measured on another device in a synchronized manner.

Second, each identified covert channel can be evaluated under different real-world settings and in presence of noise. In particular, one can evaluate how device movements affect covert channels with motion sensor as receivers, and robustness of the camera and microphone components in presence of dynamic content and noise, respectively. For all pair of components, one can evaluate robustness of covert channels in presence of background activity of the affecting components. Note that by including traces of reacting

components recorded in these real-world setups into tested samples, the same novelty-based approach can be used to identify features that are robust to these conditions.

Third, capacity of the covert channel can be evaluated depending on the amplitude of the interference produced by the affecting component's activity, if this amplitude can be controlled. In particular, to increase the covertness of the channel, an attacker would aim to not affect the utility of the reacting component, yet be able to detect and decode the transmitted signal. For example, for the reaction of camera to ultrasonic sounds, the attacker may want to cause the disturbance in the video stream that would not be visually perceived by human eyes.

Finally, other state-of-the-art methods for the novelty detection can be applied, to improve the detection capabilities and performance of our approach in multidimensional feature space. In particular, deep learning methods such as Long short-term memory (LSTM) and Autoencoders are proven to be efficient for outlier and novelty detection for time series inputs in different domains [BW20; Fer+22].

8

CONCLUSION

In this chapter, we summarize the main contributions of the thesis and outline directions for future work.

8.1 CONTRIBUTIONS

In this thesis, we have studied feasibility and applicability of physical covert channels on mobile platforms. We presented covert channels that are based on reaction of embedded sensors to electromagnetic and acoustic noise, demonstrated relevant application scenarios, and proposed a holistic approach to identify and evaluate such covert channels.

First, we presented a covert channel based on the reaction of magnetometer sensors to electromagnetic activity of other electronic components. In Chapter 2, we demonstrated that a smartphone's magnetometer can capture activity caused by the CPU and hard drive of a nearby computer. As a result, an inter-device covert channel can be constructed between these two devices. Although the distance and potential capacity of this covert channel is limited, utilizing a ubiquitous smartphone as the receiver allows an attacker to establish a covert channel without relying on dedicated equipment, extending the overall feasibility of the attack aiming to exfiltrate data from physically isolated computers over the air. In Chapter 3, we demonstrated that the reaction of the magnetometer to the CPU activity can be observed within one device, and therefore can be used to establish an intra-device covert channel. As a target application, we introduced a web-tracking scenario, where covert channel is used to break the logical isolation of the web browsing tab from native mobile applications or other web tabs, and to exfiltrate the tracking identifier from the isolated browsing session even in very restrictive browsing configurations. In Chapter 4, we further analyzed the reaction of magnetometers to the CPU activity of a smartphone, identified this reaction on numerous Android and iOS devices, and confirmed that the sensor disturbance pattern correlates with the produced CPU activity pattern. We demonstrated that this reaction can be used to establish a side-channel attack, namely to successfully infer running applications and web pages.

Second, we investigated the reaction of MEMS gyroscopes in mobile devices to resonance ultrasonic sounds, as well as reaction of MEMS accelerometers to acoustic vibrations in the sub-bass range. These covert channels can have a transmission range of up to several meters. As the target application scenario, we considered the cross-device tracking, the privacy attack aiming to link two devices to the same user, for example, for advertising purposes. Being performed without user explicit consent, such tracking can violate privacy of end users. Furthermore, we point out that these covert channels can be also applied in other application scenarios, such as web tracking, location tracking, or general data exfiltration.

Finally, in Chapter 7, we proposed to generalize covert channels that are based on physical interference between hardware components of the device, and introduced a method and an evaluation framework that allow us to automatically detect physical

interference between components that can be used to establish covert channels. This developed framework can identify physical interference between different components of mobile devices in a unified manner and on numerous devices. As a result of the evaluation, we confirmed covert channels known from related works and presented in the scope of this thesis, and identified several new potential covert channels.

Overall, our studies and results demonstrate that hardware complexity of mobile devices opens the possibility to establish covert channels, applicable to multiple application scenarios targeting both security of the computer systems and privacy of end users. We hope that the results of this work raise attention to the field of covert channels, the risks they pose in different applications scenarios, and will signify the importance of holistic approaches when identifying and evaluating covert channels.

8.2 FUTURE WORK

In this section, we discuss several research directions that further extend the scope and content of this thesis.

AUTOMATIC DETECTION OF CROSS-COMPONENT COVERT CHANNELS. Further development of automatic detection of covert channels, introduced in Chapter 7, is a natural next step expanding the scope of this thesis. Possible research directions include applying advanced methods of anomaly detection, large-scale evaluation of other hardware components and devices, applying the methodology to identify inter-device covert channels, and estimating the covert channel capacity in different realistic scenarios. We discuss these possible steps in detail in Section 7.4.

SENSOR-BASED COVERT CHANNELS IN OTHER DOMAINS. In this thesis, we have focused on feasibility and application scenarios for sensor-based covert channels on mobile devices. However, physical covert channels have been observed for other computer systems, and similar methods of constructing, detecting and evaluating covert channels can also be applied to other domains. In fact, in this work we aim to emphasize that threats of covert channels have to be considered in all complex multi-component systems that rely on numerous sensors. Modern smart home and smart city platforms are examples of such systems. They typically comprise numerous environmental sensors, as well as cameras and voice assistant devices. Existing works demonstrate attacks targeting sensors in smart home devices [Zha+17; Alr+19], and covert channels can extend existing attack surface. Physical attacks affecting cameras and microphones may be critical for security-sensitive environments that contain multiple surveillance camera or microphones. Threats of covert channels can also be explored in the automotive domain. Modern automotive architectures for smart and autonomous vehicles rely on numerous components connected over in-vehicle network, and heavily rely on sensor data. Covert channels can circumvent logical isolation between these components, or enable exfiltration of sensitive information from the vehicle to the attacker over the air.

Benevolent applications of sensor-based covert channels constitute another direction for future work. In particular, ubiquity of covert channels utilizing smartphone sensors, their unobtrusiveness, and limited transmission distance make them potentially suitable

communication channels for performing two-factor authentication (e.g., as proposed in [Far+16]) or for establishing secure device pairing (surveyed in [Fom+18]).

COVERTNESS (DETECTABILITY) OF THE CHANNELS. In this work, we considered covert communication that (only) remains unnoticed by the *end users* of the system. However, these covert channels may still be *detectable* in presence of a warden who is actively trying to detect the transmission (e.g., by analyzing the spectrum of the sensor's time series). As shown in research works [CA15], using spread-spectrum signals at low powers can improve the covertness of the channel and hide the communication even from the active detector. One reasonable direction for future work would be to extend our approach on detecting cross-component covert channels to additionally model and evaluate covertness of the channel.

GENERIC COUNTERMEASURES FOR SENSOR-BASED COVERT CHANNELS. Another important research direction is implementing countermeasures that limit the capacity of sensor-based covert channels, ideally making them impractical for target application scenarios. These countermeasures include introducing noise in sensor measurements, reducing the available signal bandwidth and power, or introducing timing variations in the sensor measurements. On possible extension of our work on automatic detection of cross-component covert channel can be evaluation of these countermeasures in a unified manner for all detected covert channels, and measuring impact of countermeasures on the utility of components.

SENSOR-BASED ATTACKS AND (IN)SECURE SENSOR FUSION. Finally, one interesting direction for future work is study of covert channels targeting sensor fusion, i.e., combined measurements from multiple sensors. It is a common practice to combine inputs from multiple sensors in order to precisely compute a certain property. For example, inputs from gyroscope, accelerometer and magnetometer can be combined to estimate device's motion or orientation (in particular, to compensate gyroscope's drift). Research works demonstrate that signal injection attacks may still affect the properties computed using the sensor fusion [Nas+18]. Therefore, one direction for future work would be to study covert channels with sensor-fusion traces as communication medium.

Another interesting direction for future work is utilizing sensor fusion as a defense against sensor-based attacks. In this case, input from other sensors can be used to either detect an attack on an individual sensor (e.g., as proposed in [Tha+20] for motion sensors), or even to prevent the attack, i.e., to correct affected sensor measurements. While methods for general sensor fusion, such as Kalman filters, are designed to improve the overall accuracy of the computed property based on the inputs from multiple sensors, they may not withstand intentional malicious interference. Modeling sensor fusion to be robust in presence of malicious inputs would make sensor systems secure by design and prevent risks of covert channels and other sensor-based attacks.

APPENDIX

A.1 INTRA-DEVICE MAGNETIC COVERT CHANNEL (CHAPTER 3)

The code below declares an invisible CSS animation that can be used to cause CPU loads in a browser without using JavaScript code.

Figure 30: CSS animation declaration example.

```
/* declare an animation, can be applied to
    several elements at once */
div.animated {
  animation-name: cpu-intense-animation;
  animation-duration: 100ms;
  animation-delay: 2s;
  animation-count: 10;
}


/* declare animated properties, setting their
    initial (0%) and end (100%) values */
@keyframes cpu-intense-animation {
  0% {
    /* transforming */
    width: 20px;
    height: 20px;

    /* moving */
    left: 0px;
    top: 0px;

    /* repainting */
    background-color: red;
    border-color: white;

    /* transforming and repainting text */
    font-size: 1px;
    font-family: Arial;

    /* making the animated element invisible */
    opacity: 0;
  }

  100% {
    /* transforming */
    width: 100px;
    height: 1000px;

    /* moving */
    left: 100px;
    top: 100px;

    /* repainting */
    background-color: green;
    border-color: black;

    /* transforming and repainting text */
    font-size: 50px;
    font-family: Arial;

    /* making the animated element invisible */
    opacity: 0.0001;
  }}
```

A.2 INTRA-DEVICE MAGNETIC SIDE CHANNEL (CHAPTER 4)

Table 24: List of mobile devices and their magnetometers not affected by their CPU activity.

Smartphone	Setup	Magnetometer
Android		
Asus ZenFone 5Z	V	AKM AK0991X
Google Pixel 2 XL	V,A	AKM AK09915
LGE LG G7 ThinQ	V,A	AKM LGE
Motorola Nexus 6	V,A	Invensense Inc.
Motorola Moto G(6) plus	V	AKM AK09918
Motorola Moto G(7) plus	V	MEMSIC MMC5603NJ
Motorola One	V	MEMSIC MMC3630KJ
OnePlus 5T	V	AKM AK09911
OnePlus 6	V	AKM AK0991X
Samsung Galaxy A6+	V	Yamaha YAS539
Samsung Galaxy A8	V	AKM AK09918
Samsung Galaxy S7 edge	V	Yamaha YAS537
Samsung Galaxy S8	V,A	AKM AK09916C
Samsung Galaxy S8+	V,A	AKM AK09916C
Samsung Galaxy S9	V,A	AKM AK09916C
Samsung Galaxy S10+	V,A	AKM AK09918C
Samsung Galaxy Tab S2	V	Yamaha
Samsung Galaxy Tab S3	V,A	AKM AK09916
Samsung Galaxy Tab S4	V	AKM AK09918
Sony Xperia 10 Plus	V	GlobalMEMS GMC306
Sony Xperia XZ2 Compact	V	AKM AK0991X
iOS		
iPad Air 2019	V	Unknown
iPad Mini 4	V	Unknown
iPad Pro	V	AKM AK8789
iPad Pro 9.7	V	Unknown

Table 25: Full list of mobile devices on which sensor measurements correlate with CPU activity.

Smartphone	Setup[a]	Magnetometer[c]	Correlation		SNR,
			Pattern	CPU[b]	dB
Android					
Essential Products PH-1	V	AKM AK09915	0.83	—	10.2
Google Pixel	V,A,L	AKM AK09915	0.86	0.89	14.7
Google Pixel 2	V,A,L	AKM AK09915	0.78	—	10.8
Google Pixel 3	V,A	STMicro LIS2MDL	0.90	—	14.2
Google Pixel 3 XL	V,A	STMicro LIS2MDL	0.91	—	15.4
Google Pixel C	V	Google CROSEC	0.91	—	27.4
Google Pixel XL	V,A	AKM AK09915	0.83	0.95	12.2
HTC U Ultra	V	AKM AK09915	0.95	0.96	28.6
HTC U11	V,A	AKM AK09915	0.50	—	−9.7
HTC U12+	V	AKM AK09915	0.60	—	7.1
Huawei Honor View 10	V	AKM AK09918	0.81	—	10.8
Huawei Mate 20 Pro	V	AKM	0.81	—	20.1
Huawei Mate 10 Pro	V	AKM	0.87	—	13.0
Huawei Nexus 6P	V	Bosch BMM150	0.85	0.94	14.5
Huawei P10	V	AKM	0.84	0.89	15.8
Huawei P20 Pro	V	AKM	0.60	—	3.4
Huawei P30 Pro	V	AKM AK09918	0.54	—	4.2
LG G6	V	AKM LGE	0.86	0.88	12.1
LG Nexus 5X	V,L	Bosch BMM150	0.88	0.93	15.5
LG V30	V	AKM LGE	0.93	0.96	22.6
Motorola Moto X(4)	V	MEMSIC MMC3630KJ	0.82	—	4.2
Motorola Moto g(6)	V	AKM AK09918	0.54	—	2.9
Motorola Moto Z3 Play	V	AKM AK09915	0.79	—	13.4
OnePlus 3	V	MEMSIC MMC3416PJ	0.92	0.95	14.7
OnePlus OnePlus 6T	V	AKM AK0991X	0.40	—	−0.7
Samsung Galaxy A7	V	Yamaha YAS539	0.84	—	13.8
Samsung Galaxy Note 8	V,A	AKM AK09916C	0.91	0.95	16.3
Samsung Galaxy Note 9	V,A	AKM AK09918C	0.52	—	4.1
Samsung Galaxy S7	V	Yamaha YAS537	0.73	—	4.5
Samsung Galaxy S9+	V,A	AKM AK09916C	0.55	—	4.2
Samsung Galaxy S10E	V,A	AKM AK09918C	0.77	—	12.3
Samsung Galaxy S10	V,A	AKM AK09918C	0.65	—	7.9
Samsung Galaxy XCover4	V	AKM AK09916C	0.89	0.72	−1.6
Sony Xperia XZ2	V	AKM AK0991X	0.40	—	−3.0
Sony Xperia XZ3	V	AKM AK0991X	0.49	—	1.7
Sony Xperia 10 Plus	V	GlobalMEMS GMC306	0.78	—	8.8
Xiaomi Mi A1	V	AKM AK09918	0.82	—	11.0
Xiaomi Mi A2	V	AKM AK09918	0.79	—	9.7
iOS					
iPad Pro 12.9	V,A	AKM AK8789	0.93	0.63	16.3
iPad Pro 11	V,A	AKM AK8789	0.86	0.58	12.0
iPad Air 2	V,A	Unknown	0.84	0.42	13.0
iPad Mini 3	V	Unknown	0.95	0.96	16.8
iPhone 5S	V,A	AKM AK8963	0.89	0.80	12.1
iPhone SE	V	Alps HSCDTD007	0.91	0.87	19.2
iPhone 6	A	AKM AK8963	0.70	0.59	8.4
iPhone 6S	V,A,L	Alps HSCDTD007	0.81	0.81	20.3
iPhone 6S Plus	V	Alps HSCDTD007	0.93	0.91	22.9
iPhone 7	V,A,L	Alps HSCDTD008A	0.89	0.85	11.0
iPhone 7 Plus	V,A	Alps HSCDTD008A	0.82	0.76	9.6
iPhone 8	V,A	Unknown	0.85	0.88	13.4
iPhone 8 Plus	V,A	Alps e-Compass	0.87	0.81	12.0
iPhone X	V,A	Unknown	0.77	0.74	22.5
iPhone XR	V,A	Unknown	0.88	0.86	16.9
iPhone XS	V,A	Unknown	0.75	0.72	12.1
iPhone XS Max	V,A	Unknown	0.76	0.74	11.7

[a] V — Visual Studio App Center; A — AWS Device Farm; L — lab

[b] CPU utilization is available only on Android $\leqslant 7$ (over */proc/stat*) and on iOS (over *host_processor_info*).

[c] For Android devices, as provided by the *Sensor* API. For iOS devices, information is taken from public online resources.

Table 26: Classification results for websites and applications.

Website	Precision	Recall	F1 score
360.cn	0.87	0.83	0.85
accuweather.com	0.82	0.96	0.89
aliexpress.com	1.00	1.00	1.00
alipay.com	0.87	0.87	0.87
amazon.com	0.92	0.97	0.95
ampproject.org	0.86	0.90	0.88
apple.com	0.83	0.85	0.84
baidu.com	0.82	0.91	0.86
bing.com	0.88	1.00	0.93
blogspot.com	0.95	0.93	0.94
csdn.net	0.92	0.80	0.86
ebay.com	0.98	0.92	0.95
facebook.com	0.86	0.78	0.82
google.com	0.74	0.89	0.81
imdb.com	0.98	1.00	0.99
instagram.com	0.96	0.94	0.95
jd.com	0.81	0.96	0.88
linkedin.com	1.00	0.98	0.99
live.com	0.88	0.84	0.86
mail.ru	1.00	1.00	1.00
mi.com	0.98	0.82	0.89
microsoft.com	0.95	0.89	0.92
msn.com	1.00	0.90	0.95
naver.com	1.00	0.98	0.99
netflix.com	0.87	0.79	0.83
office.com	0.81	0.90	0.85
ok.ru	0.95	0.86	0.90
pinterest.com	1.00	0.91	0.96
qq.com	0.83	0.98	0.90
reddit.com	0.93	0.87	0.90
samsung.com	0.98	1.00	0.99
sina.com.cn	0.86	0.87	0.86
sm.cn	0.87	0.81	0.84
sogou.com	0.70	0.92	0.80
sohu.com	0.95	0.95	0.95
spotify.com	0.93	0.89	0.91
store.google.com	0.98	0.89	0.93
taobao.com	0.93	0.86	0.89
tmall.com	0.98	0.96	0.97
twitch.tv	0.94	0.96	0.95
twitter.com	1.00	0.98	0.99
vk.com	0.94	0.87	0.90
weibo.com	0.77	0.79	0.78
whatsapp.com	0.86	0.79	0.83
wikipedia.org	0.91	0.82	0.87
yahoo.co.jp	0.96	1.00	0.98
yahoo.com	0.87	0.97	0.92
yandex.ru	0.97	0.97	0.97
yidianzixun.com	0.89	0.89	0.89
youtube.com	0.89	0.89	0.89
Average	0.91	0.91	0.91

Application	Precision	Recall	F1 score
com.UCMobile.intl	0.92	0.85	0.88
com.airbnb.android	1.00	1.00	1.00
com.amazon.mShop.android.shopping	1.00	0.97	0.99
com.android.chrome	0.96	0.96	0.96
com.android.vending	0.97	0.94	0.95
com.booking	0.80	0.97	0.88
com.cleanmaster.mguard	0.91	0.94	0.92
com.cleanmaster.security	0.88	0.98	0.93
com.cmplay.tiles2	0.81	0.90	0.85
com.contextlogic.wish	0.93	0.93	0.93
com.dianxinos.dxbs	1.00	0.98	0.99
com.dianxinos.optimizer.duplay	0.83	0.98	0.90
com.dts.freefireth	1.00	0.94	0.97
com.etermax.preguntados.lite	0.91	1.00	0.95
com.facebook.katana	0.88	0.88	0.88
com.facebook.lite	0.72	0.79	0.76
com.facebook.orca	0.98	0.95	0.97
com.fingersoft.hillclimb	0.96	0.98	0.97
com.firsttouchgames.dls3	0.95	0.54	0.69
com.fungames.sniper3d	0.70	0.78	0.74
com.gameloft.android.ANMP.GloftA8HM	0.89	1.00	0.94
com.gameloft.android.ANMP.GloftDMHM	0.81	0.60	0.69
com.google.android.GoogleCamera	1.00	0.98	0.99
com.google.android.apps.maps	0.93	0.91	0.92
com.google.android.apps.photos	0.96	1.00	0.98
com.google.android.apps.translate	1.00	0.92	0.96
com.google.android.calendar	0.97	0.90	0.94
com.google.android.gm	0.92	0.96	0.94
com.google.android.googlequicksearchbox	0.91	0.93	0.92
com.google.android.play.games	0.90	1.00	0.95
com.google.android.youtube	0.88	0.95	0.91
com.imangi.templerun2	0.70	0.74	0.72
com.instagram.android	0.92	0.90	0.91
com.kiloo.subwaysurf	1.00	0.49	0.66
com.king.candycrushsaga	0.57	0.95	0.71
com.king.farmheroessaga	0.98	0.98	0.98
com.lenovo.anyshare.gps	0.97	0.95	0.96
com.linkedin.android	0.85	0.95	0.90
com.miniclip.eightballpool	0.91	0.96	0.93
com.mobile.legends	0.93	1.00	0.96
com.mxtech.videoplayer.ad	0.96	0.84	0.90
com.nekki.shadowfight	1.00	0.93	0.96
com.ngame.allstar.eu	1.00	0.98	0.99
com.outfit7.mytalkingangelafree	0.88	0.70	0.78
com.outfit7.mytalkingtomfree	0.90	0.91	0.91
com.paypal.android.p2pmobile	0.98	1.00	0.99
com.picsart.studio	0.85	0.83	0.84
com.qihoo.security	0.91	0.98	0.94
com.quvideo.xiaoying	0.98	0.94	0.96
com.roidapp.photogrid	0.94	0.96	0.95
com.skype.raider	1.00	0.85	0.92
com.snapchat.android	0.98	1.00	0.99
com.spotify.music	0.92	0.90	0.91
com.supercell.clashofclans	0.70	0.82	0.75
com.supercell.clashroyale	0.98	0.96	0.97
com.supercell.hayday	0.78	0.90	0.84
com.surpax.ledflashlight.panel	0.97	0.66	0.79
com.tencent.ig	0.92	0.58	0.71
com.tencent.mm	0.92	1.00	0.96
com.tripadvisor.tripadvisor	1.00	1.00	1.00
com.twitter.android	0.95	0.74	0.83
com.waze	0.89	0.98	0.93
com.whatsapp	0.91	0.94	0.93
com.zhiliaoapp.musically	0.73	0.92	0.81
jp.naver.line.android	1.00	0.92	0.96
me.pou.app	0.84	0.97	0.90
net.zedge.android	0.65	0.82	0.73
Average	0.90	0.90	0.90

BIBLIOGRAPHY

[Aca+14] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. "The Web Never Forgets: Persistent Tracking Mechanisms in the Wild." In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2014.

[Agr+02] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. "The EM Side-Channel(s)." In: *Cryptographic Hardware and Embedded Systems - 4th International Workshop (CHES)*. 2002.

[Ale18] *Alexa Internet, Inc. Alexa Top 500 Global Sites*. (Visited on 11/23/2018). https://www.alexa.com/topsites.

[Alr+19] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. "SoK: Security Evaluation of Home-Based IoT Deployments." In: *IEEE Symposium on Security and Privacy (S&P)*. 2019.

[Ana+21] Nikolaos Athanasios Anagnostopoulos, Yufan Fan, Markus Heinrich, Nikolay Matyunin, Dominik Püllen, Philipp Muth, Christian Hatzfeld, Markus Rosenstihl, Tolga Arul, and Stefan Katzenbeisser. "Low-Temperature Attacks against Digital Electronics: A Challenge for the Security of Superconducting Modules in High-Speed Magnetic Levitation (MagLev) Trains." In: *IEEE 14th Workshop on Low Temperature Electronics (WOLTE)*. 2021.

[AS18] Abhishek Anand and Nitesh Saxena. "Speechless: Analyzing the Threat to Speech Privacy from Smartphone Motion Sensors." In: *IEEE Symposium on Security and Privacy (S&P)*. 2018.

[Anda] *Android 9 Behaviour Changes: All Apps*. (Visited on 11/24/2018). https://developer.android.com/about/versions/oreo/background.

[Andb] *Android Debug Bridge (Adb) | Android Developers*. (Visited on 11/28/2018). https://developer.android.com/studio/command-line/adb.

[And18a] *Android Market History Data and Ranklists | 2011 - 2018*. (Visited on 11/23/2018). https://www.androidrank.org/.

[And18b] *Android Open Source Project: Bug 23310674: Enable Hidepid=2 on Proc*. (Visited on 11/28/2018). https://android-review.googlesource.com/c/platform/system/core/+/181345.

[Ani20] *Animations and Performance | Web Fundamentals*. (Visited on 02/11/2022). https://developers.google.com/web/fundamentals/design-and-ux/animations/animations-and-performance.

[App18a] Apple corp. *iOS Security Guide*. (Visited on 11/22/2018). https://source.android.com/security/app-sandbox.

[App18b] *Apple Developer Documentation | Core Motion Framework*. (Visited on 10/23/2018). https://www.techspot.com/news/66899-mobile-web-browsing-becomes-more-popular-than-desktop.html.

[App20] *Apple Developer Documentation | Preparing Your UI to Run in the Background*. (Visited on 07/22/2020). https://developer.apple.com/documentation/uikit/app_and_environment/scenes/preparing_your_ui_to_run_in_the_background.

[App18c] *Application Sandbox | Android Open Source Project*. (Visited on 11/22/2018). https://source.android.com/security/app-sandbox.

[AQW16] Daniel Arp, Erwin Quiring, and Christian Wressnegger. *Bat in the Mobile : A Study on Ultrasonic Device Tracking*. Technical report. Technische Universität Braunschweig, 2016.

[Arp+17] Daniel Arp, Erwin Quiring, Christian Wressnegger, and Konrad Rieck. "Privacy Threats through Ultrasonic Side Channels on Mobile Devices." In: *IEEE European Symposium on Security and Privacy (EuroS&P)*. 2017.

[AWS20] *AWS Device Farm - Amazon Web Services*. (Visited on 07/22/2020). https://aws.amazon.com/device-farm/.

[Ba+20] Zhongjie Ba, Tianhang Zheng, Xinyu Zhang, Zhan Qin, Baochun Li, Xue Liu, and Kui Ren. "Learning-Based Practical Smartphone Eavesdropping with Built-in Accelerometer." In: *Annual Network and Distributed System Security Symposium (NDSS)*. 2020.

[Bac18] *Background Execution Limits | Android Developers*. (Visited on 11/24/2018). https://developer.android.com/about/versions/oreo/background.

[Bar+20] Marília Barandas, Duarte Folgado, Letícia Fernandes, Sara Santos, Mariana Abreu, Patrícia J. Bota, Hui Liu, Tanja Schultz, and Hugo Gamboa. "TSFEL: Time Series Feature Extraction Library." In: *SoftwareX* 11 (2020).

[Bar53] Ronald Hugh Barker. "Group Sysnchronizing of Binary Digital Systems." In: *Communication Theory* (1953).

[BKS15] Sebastian Biedermann, Stefan Katzenbeisser, and Jakub Szefer. "Hard Drive Side-Channel Attacks Using Smartphone Magnetic Field Sensors." In: *19th International Conference on Financial Cryptography and Data Security (FC)*. 2015.

[BNN17] Kenneth Block, Sashank Narain, and Guevara Noubir. "An Autonomic and Permissionless Android Covert Channel." In: *10th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. 2017.

[BN18] Kenneth Block and Guevara Noubir. "My Magnetometer Is Telling You Where I've Been?: A Mobile Device Permissionless Location Attack." In: *11th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. 2018.

[Boj+14] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. "Mobile Device Identification via Sensor Fingerprinting." 2014. arXiv: 1408.1416 [quant-ph].

[BW20] Mohammad Braei and Sebastian Wagner. "Anomaly Detection in Univariate Time-series: A Survey on the State-of-the-Art." In: (2020). arXiv: 2004.00433 [cs.LG].

[Bre01] Leo Breiman. "Random Forests." In: *Machine Learning* 45.1 (2001).

[Bre+00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. "LOF: Identifying Density-Based Local Outliers." In: *ACM International Conference on Management of Data (SIGMOD)*. 2000.

[Bre18] Thomas Brewster. *Meet the Ultrasonic Tracking Company Privacy Activists Are Terrified Of*. (Visited on 07/17/2018). https://www.counterpointresearch.com/almost-half-of-smartphone-users-spend-more-than-5-hours-a-day-on-their-mobile-device/.

[Bro+17] Justin Brookman, Phoebe Rouge, Aaron Alva, and Christina Yeung. "Cross-Device Tracking: Measurement and Disclosures." In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.2 (2017).

[Cai+12] Yongyao Cai, Yang Zhao, Xianfeng Ding, and James Fennelly. "Magnetometer Basics for Mobile Phone Applications." In: *Electron. Prod* 54.2 (2012).

[Cal+15a] Chris Calabrese, Katherine L. McInnis, Gautam S. Hans, and Greg Norcie. *Comments for November 2015 Workshop on Cross-device Tracking*. Report. Center for Democracy & Technology, 2015, pp. 1–11.

[Cal+15b] Robert Callan, Nina Popovic, Alenka Zajić, and Milos Prvulovic. "A New Approach for Measuring Electromagnetic Side-Channel Energy Available to the Attacker in Modern Processor-Memory Systems." In: *9th European Conference on Antennas and Propagation (EuCAP)*. 2015.

[CZP14] Robert Callan, Alenka Zajic, and Milos Prvulovic. "A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events." In: *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2014.

[CA14] Brent Carrara and Carlisle Adams. "On Acoustic Covert Channels between Air-Gapped Systems." In: *International Symposium on Foundations and Practice of Security*. 2014.

[CA15] Brent Carrara and Carlisle Adams. "On Characterizing and Measuring Out-of-Band Covert Channels." In: *3rd ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec)*. New York, NY, USA, 2015.

[CA16] Brent Carrara and Carlisle Adams. "Out-of-Band Covert Channels - A Survey." In: *ACM Computing Surveys (CSUR)* 49.2 (2016).

[CH10] Aaron Carroll and Gernot Heiser. "An Analysis of Power Consumption in a Smartphone." In: *USENIX Annual Technical Conference*. 2010.

[Cas+07] Simon Castro, Robert Dean, Grant Roth, George T. Flowers, and Brian Grantham. "Influence of Acoustic Noise on the Dynamic Performance of MEMS Gyroscopes." In: *ASME International Mechanical Engineering Congress and Exposition*. 2007.

[Cha+14] Swarup Chandra, Zhiqiang Lin, Ashish Kundu, and Latifur Khan. "Towards a Systematic Study of the Covert Channel Attacks in Smartphones." In: *International Conference on Security and Privacy in Communication Systems*. 2014.

[CQM14] Qi Alfred Chen, Zhiyun Qian, and Zhuoqing Morley Mao. "Peeking into Your App without Actually Seeing It: UI State Inference and Novel Android Attacks." In: *23rd USENIX Security Symposium*. 2014.

[Che+13] Xiang Chen, Yiran Chen, Zhan Ma, and Felix C. A. Fernandes. "How Is Energy Consumed in Smartphone Display Applications?" In: *14th Workshop on Mobile Computing Systems and Applications (HotMobile)*. 2013.

[Che+17] Yimin Chen, Xiaocong Jin, Jingchao Sun, Rui Zhang, and Yanchao Zhang. "POWERFUL: Mobile App Fingerprinting via Power Analysis." In: *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. 2017.

[Che+19] Yushi Cheng, Xiaoyu Ji, Wenyuan Xu, Hao Pan, Zhuangdi Zhu, Chuang-Wen You, Yi-Chao Chen, and Lili Qiu. "MagAttack: Guessing Application Launching and Operation via Smartphone." In: *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2019.

[Chr20] *Chromium Blog: Reducing Power Consumption for Background Tabs*. (Visited on 07/25/2020).
https://blog.chromium.org/2017/03/reducing-power-consumption-for.html.

[Cla+13] Shane S. Clark, Hossen Mustafa, Benjamin Ransford, Jacob Sorber, Kevin Fu, and Wenyuan Xu. "Current Events: Identifying Webpages by Tapping the Electrical Outlet." In: *18th European Symposium on Research in Computer Security (ESORICS)*. 2013.

[CSH15] Jiska Classen, Matthias Schulz, and Matthias Hollick. "Practical Covert Channels for WiFi Systems." In: *IEEE Conference on Communications and Network Security (CNS)*. 2015.

[Com20] *Comparative Examples of Noise Levels - IAC Acoustics*. (Visited on 07/19/2020).
https://www.iacacoustics.com/blog-full/comparative-examples-of-noise-levels.html.

[CSS] *CSS and JavaScript Animation Performance - Web Performance | MDN*. (Visited on 02/11/2022).
https://developer.mozilla.org/en-US/docs/Web/Performance/CSS_JavaScript_animation_performance.

[DBC14] Anupam Das, Nikita Borisov, and Matthew Caesar. "Do You Hear What i Hear?: Fingerprinting Smart Devices through Embedded Acoustic Components." In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2014.

[DJ89] Gary Davis and Ralph Jones. *The Sound Reinforcement Handbook*. Hal Leonard Corporation, 1989.

[Dea+07] Robert N. Dean, George T. Flowers, Alan Scottedward Hodel, Grant Roth, Simon Castro, Ran Zhou, Alfonso Moreira, Anwar Ahmed, Rifki Rifki, Brian E. Grantham, et al. "On the Degradation of MEMS Gyroscope Performance in the Presence of High Power Acoustic Noise." In: *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium On*. 2007.

[Den+21] Shuwen Deng, Nikolay Matyunin, Wenjie Xiong, Stefan Katzenbeisser, and Jakub Szefer. "Evaluation of Cache Attacks on Arm Processors and Secure Caches." In: *IEEE Transactions on Computers* (2021).

[Des14] Luke Deshotels. "Inaudible Sound as a Covert Channel in Mobile Devices." In: *8th USENIX Workshop on Offensive Technologies (WOOT)*. 2014.

[Dev18] *DeviceMotionEvent | Mozilla*. (Visited on 10/23/2018). https://www.techspot.com/news/66899-mobile-web-browsing-becomes-more-popular-than-desktop.html.

[Dey+14] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. "AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable." In: *Annual Network and Distributed System Security Symposium (NDSS)*. 2014.

[Dia+16] Wenrui Diao, Xiangyu Liu, Zhou Li, and Kehuan Zhang. "No Pardon for the Interruption: New Inference Attacks on Android through Interrupt Timing Analysis." In: *IEEE Symposium on Security and Privacy (S&P)*. 2016.

[Dic05] Vance Dickason. *Loudspeaker Design Cookbook*. Audio Amateur, Inc., 2005.

[Dis18] *Distribution Dashboard | Android Developers*. (Visited on 11/24/2018). https://developer.android.com/about/dashboards/.

[Eck10] Peter Eckersley. "How Unique Is Your Web Browser?" In: *Proceedings on Privacy Enhancing Technologies (PoPETs)*. Lecture Notes in Computer Science (2010). Ed. by Mikhail J. Atallah and Nicholas J. Hopper.

[Eng18] Eric Enge. *Mobile vs Desktop Usage in 2018: Mobile Takes the Lead*. (Visited on 07/17/2018). https://www.stonetemple.com/mobile-vs-desktop-usage-study/.

[Far+16] Benyamin Farshteindiker, Nir Hasidim, Asaf Grosz, and Yossi Oren. "How to Phone Home with Someone Else's Phone: Information Exfiltration Using Intentional Sound Noise on Gyroscopic Sensors." In: *10th USENIX Workshop on Offensive Technologies (WOOT)*. 2016.

[Fer+22] Tharindu Fernando, Harshala Gammulle, Simon Denman, Sridha Sridharan, and Clinton Fookes. "Deep Learning for Medical Anomaly Detection - A Survey." In: *ACM Computing Surveys* 54.7 (2022).

[Fom+18] Mikhail Fomichev, Flor Álvarez, Daniel Steinmetzer, Paul Gardner-Stephen, and Matthias Hollick. "Survey and Systematization of Secure Device Pairing." In: *IEEE Communications Surveys & Tutorials* 20.1 (2018).

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. "Electromagnetic Analysis: Concrete Results." In: *Cryptographic Hardware and Embedded Systems - 3rd International Workshop (CHES)*. Generators. 2001.

[Gaz+22] Matthias Gazzari, Annemarie Mattmann, Max Maass, and Matthias Hollick. "My(o) Armband Leaks Passwords: An EMG and IMU Based Keylogging Side-Channel Attack." In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)* 5.4 (2022).

[Gen+15] Daniel Genkin, Lev Pachmanov, Itamar Pipman, and Eran Tromer. "Stealing Keys from PCs Using a Radio: Cheap Electromagnetic Attacks on Windowed Exponentiation." In: *Cryptographic Hardware and Embedded Systems - 17th International Workshop (CHES)*. 2015.

[GR19] Ilias Giechaskiel and Kasper Rasmussen. "Taxonomy and Challenges of Out-of-Band Signal Injection Attacks and Defenses." In: *IEEE Communications Surveys & Tutorials* 22.1 (2019).

[Goo18] *Google Issue Tracker: 37140047: Android O Prevents Access to /Proc/Stat*. (Visited on 11/28/2018). https://issuetracker.google.com/issues/37140047.

[GLB15] Mario Guggenberger, Mathias Lux, and László Böszörményi. "An Analysis of Time Drift in Hand-Held Recording Devices." In: *21th International Conference on Multimedia Modeling (MMM)*. 2015.

[Gül+17] Berk Gülmezoglu, Andreas Zankl, Thomas Eisenbarth, and Berk Sunar. "PerfWeb: How to Violate Web Privacy with Hardware Performance Events." In: *22nd European Symposium on Research in Computer Security (ESORICS)*. 2017.

[Gur21a] Mordechai Guri. "GAIROSCOPE: Leaking Data from Air-Gapped Computers to Nearby Smartphones using Speakers-to-Gyro Communication." In: *18th International Conference on Privacy, Security and Trust (PST)*. 2021.

[Gur21b] Mordechai Guri. "MAGNETO: Covert Channel between Air-Gapped Systems and Nearby Smartphones via CPU-generated Magnetic Fields." In: *Future Generation Computer Systems* 115 (2021).

[Gur+15a] Mordechai Guri, Assaf Kachlon, Ofer Hasson, Gabi Kedma, Yisroel Mirsky, and Yuval Elovici. "GSMem: Data Exfiltration from Air-Gapped Computers over GSM Frequencies." In: *24th USENIX Security Symposium*. 2015.

[Gur+14] Mordechai Guri, Gabi Kedma, Assaf Kachlon, and Yuval Elovici. "AirHopper: Bridging the Air-Gap between Isolated Networks and Mobile Phones Using Radio Frequencies." In: *9th International Conference on Malicious and Unwanted Software (MALWARE)*. 2014.

[Gur+15b] Mordechai Guri, Matan Monitz, Yisroel Mirski, and Yuval Elovici. "BitWhisper: Covert Signaling Channel between Air-Gapped Computers Using Thermal Manipulations." In: *IEEE 28th Computer Security Foundations Symposium (CSF)*. 2015.

[Gur+18] Mordechai Guri, Boris Zadov, Andrey Daidakulov, and Yuval Elovici. "xLED: Covert Data Exfiltration from Air-Gapped Networks via Switch and Router LEDs." In: *16th Annual Conference on Privacy, Security and Trust (PST)*. 2018.

[GZE17] Mordechai Guri, Boris Zadov, and Yuval Elovici. "LED-it-GO: Leaking (A Lot of) Data from Air-Gapped Computers via the (Small) Hard Drive LED." In: *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. 2017.

[GZE20] Mordechai Guri, Boris Zadov, and Yuval Elovici. "ODINI: Escaping Sensitive Data From Faraday-Caged, Air-Gapped Computers via Magnetic Fields." In: *IEEE Transactions on Information Forensics and Security* 15 (2020).

[AIN14] Ahmed Al-Haiqi, Mahamod Ismail, and Rosdiadee Nordin. "A New Sensors-Based Covert Channel on Android." In: *The Scientific World Journal* 2014 (2014).

[HSM94] Petter Hallmo, Arne Sundby, and Iain W. S. Mair. "Extended High-Frequency Audiometry: Air-and Bone-Conduction Thresholds, Age and Gender Variations." In: *Scandinavian audiology* 23.3 (1994).

[Han+12] Jun Han, Emmanuel Owusu, Le T. Nguyen, Adrian Perrig, and Joy Zhang. "ACComplice: Location Inference Using Accelerometers on Smartphones." In: *4th International Conference on Communication Systems and Networks (COMSNETS)*. 2012.

[HG13] Michael Hanspach and Michael Goetz. "On Covert Acoustical Mesh Networks in Air." In: *Journal of Communications* 8.11 (2013). arXiv: 1406.1213.

[HG14] Michael Hanspach and Michael Goetz. "Recent Developments in Covert Acoustical Communications." In: *Sicherheit* (2014).

[Has+13] Ragib Hasan, Nitesh Saxena, Tzipora Halevi, Shams Zawoad, and Dustin Rinehart. "Sensing-Enabled Channels for Hard-to-Detect Command and Control of Mobile Devices." In: *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2013.

[Hir20] *Hi-Res Audio | High Resolution Audio for Best Sound Quality | Sony US*. (Visited on 07/19/2020). https://www.sony.com/electronics/hi-res-audio.

[Hot33] Harold Hotelling. "Analysis of a Complex of Statistical Variables into Principal Components." In: *Journal of Educational Psychology* 24.6 (1933).

[Hul+10] Gauthier Hulot, Christopher Finlay, Catherine Constable, Nils Olsen, and Mioara Mandea. "The Magnetic Field of Planet Earth." In: *Space Science Reviews* 152.1-4 (2010).

[Hus+16] Muzammil Hussain, Ahmed Al-Haiqi, Aws Alaa Zaidan, Bilal Bahaa Zaidan, M. L. Mat Kiah, Nor Badrul Anuar, and Mohamed Abdulnabi. "The Rise of Keyloggers on Smartphones: A Survey and Insight into Motion-Based Tap Inference Attacks." In: *Pervasive and Mobile Computing* 25 (2016).

[ISO22603] *International Standard ISO 226: 2003: Normal Equal-Loudness-Level*. International Organization for Standardization, Geneva, Switzerland. https://www.iso.org/standard/34222.html.

[Int20] *Intervention: Throttle Expensive Background Timers - Chrome Platform Status*. (Visited on 07/25/2020). https://www.chromestatus.com/feature/6172836527865856.

[JS12] Suman Jana and Vitaly Shmatikov. "Memento: Learning Secrets from Process Footprints." In: *IEEE Symposium on Security and Privacy (S&P)*. 2012.

[Jap20] *Japan Audio Society | Status of Hi-Res AUDIO Logo*. (Visited on 07/19/2020). https://www.jas-audio.or.jp/english/hi-res-logo-en/use-situation-en.

[Ji+21] Xiaoyu Ji, Yushi Cheng, Yuepeng Zhang, Kai Wang, Chen Yan, Wenyuan Xu, and Kevin Fu. "Poltergeist: Acoustic Adversarial Machine Learning against Cameras and Computer Vision." In: *IEEE Symposium on Security and Privacy (S&P)*. 2021.

[Jon10] Willie D. Jones. "A Compass in Every Smartphone." In: *IEEE Spectrum* 47.2 (2010).

[Juá+14] Marc Juárez, Sadia Afroz, Gunes Acar, Claudia Díaz, and Rachel Greenstadt. "A Critical Evaluation of Website Fingerprinting Attacks." In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2014.

[Kaa09] Ville Kaajakari. "Practical MEMS — Design of Microsystems, Accelerometers, Gyroscopes." In: *MEMS, optical MEMS and microfluidic systems* (2009).

[Kam10] Samy Kamkar. *Evercookie - Virtually Irrevocable Persistent Cookies*. (Visited on 07/25/2020).
https://samy.pl/evercookie/.

[Koc+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. "Spectre Attacks: Exploiting Speculative Execution." In: *IEEE Symposium on Security and Privacy (S&P)*. 2019.

[Kuh02] Markus G. Kuhn. "Compromising Emanations: Eavesdropping Risks of Computer Displays." Citeseer, 2002.

[KA98] Markus G. Kuhn and Ross J. Anderson. "Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations." In: *Information Hiding, 2nd International Workshop*. 1998.

[Kul19] Kristian Dietmar Kullmann. "A Framework for Systematic Evaluation of Sensor-Based Physical Side-Channel Attacks on Apple iOS Operating System." Darmstadt University of Technology, Germany, 2019.

[Kun+13] Denis Foo Kune, John Backes, Shane S. Clark, Daniel Kramer, Matthew Reynolds, Kevin Fu, Yongdae Kim, and Wenyuan Xu. "Ghost Talk: Mitigating EMI Signal Injection Attacks against Analog Sensors." In: *IEEE Symposium on Security and Privacy (S&P)*. 2013.

[Lam73] Butler W. Lampson. "A Note on the Confinement Problem." In: *Communications of the ACM* 16.10 (1973).

[LRB16] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. "Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints." In: *IEEE Symposium on Security and Privacy (S&P)*. 2016.

[Lea00] Robert H. Leary. "Global Optimization on Funneling Landscapes." In: *Journal of Global Optimization* 18.4 (2000).

[Lee+12] Jungmee Lee, Sumitrajit Dhar, Rebekah Abel, Renee Banakis, Evan Grolley, Jungwha Lee, Steven Zecker, and Jonathan Siegel. "Behavioral Hearing Thresholds between 0.125 and 20 kHz Using Depth-Compensated Ear Simulator Calibration." In: *Ear and Hearing* 33.3 (2012).

[Lif+18] Pavel Lifshits, Roni Forte, Yedid Hoshen, Matt Halpern, Manuel Philipose, Mohit Tiwari, and Mark Silberstein. "Power to Peep-All: Inference Attacks by Malicious Batteries on Mobile Devices." In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2018.4 (2018).

[Lip+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. "Meltdown: Reading Kernel Memory from User Space." In: *27th USENIX Security Symposium*. 2018.

[Lu18] Tina Lu. *Almost Half of Smartphone Users Spend More than 5 Hours a Day on Their Mobile Device - Counterpoint Research*. (Visited on 07/17/2018). https://www.counterpointresearch.com/almost-half-of-smartphone-users-spend-more-than-5-hours-a-day-on-their-mobile-device/.

[MJ19] Anindya Maiti and Murtuza Jadliwala. "Light Ears: Information Leakage via Smart Lights." In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)* 3.3 (2019).

[Mai+15] Anindya Maiti, Murtuza Jadliwala, Jibo He, and Igor Bilogrevic. "(Smart)Watch Your Taps: Side-Channel Keystroke Inference Attacks Using Smartwatches." In: *ACM International Symposium on Wearable Computers (ISWC)*. 2015.

[Maj18] *Majestic Million - Majestic*. (Visited on 11/29/2018). https://majestic.com/reports/majestic-million.

[Mar+12] Claudio Marforio, Hubert Ritzdorf, Aurélien Francillon, and Srdjan Capkun. "Analysis of the Communication between Colluding Applications on Modern Smartphones." In: *28th Annual Computer Security Applications Conference (ACSAC)*. 2012.

[Mas+15] Ramya Jayaram Masti, Devendra Rai, Aanjhan Ranganathan, Christian Müller, Lothar Thiele, and Srdjan Capkun. "Thermal Covert Channels on Multi-core Platforms." In: *24th USENIX Security Symposium*. 2015.

[Mat+18] Nikolay Matyunin, Nikolaos Athanasios Anagnostopoulos, Spyros Boukoros, Markus Heinrich, André Schaller, Maksim Kolinichenko, and Stefan Katzenbeisser. "Tracking Private Browsing Sessions Using CPU-based Covert Channels." In: *11th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. 2018.

[Mat+16] Nikolay Matyunin, Jakub Szefer, Sebastian Biedermann, and Stefan Katzenbeisser. "Covert Channels Using Mobile Device's Magnetic Field Sensors." In: *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2016.

[MSK18] Nikolay Matyunin, Jakub Szefer, and Stefan Katzenbeisser. "Zero-Permission Acoustic Cross-Device Tracking." In: *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2018.

[Mat+19] Nikolay Matyunin, Yujue Wang, Tolga Arul, Kristian Kullmann, Jakub Szefer, and Stefan Katzenbeisser. "MagneticSpy: Exploiting Magnetometer in Mobile Devices for Website and Application Fingerprinting." In: *18th Annual ACM Workshop on Privacy in the Electronic Society (WPES)*. 2019.

[MWK19] Nikolay Matyunin, Yujue Wang, and Stefan Katzenbeisser. "Vibrational Covert Channels Using Low-Frequency Acoustic Signals." In: *ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec)*. 2019.

[Mav+17] Vasilios Mavroudis, Shuang Hao, Yanick Fratantonio, Federico Maggi, Christopher Kruegel, and Giovanni Vigna. "On the Privacy and Security of the Ultrasound Ecosystem." In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.2 (2017).

[Meh+18] Maryam Mehrnezhad, Ehsan Toreini, Siamak F. Shahandashti, and Feng Hao. "Stealing PINs via Mobile Sensors: Actual Risk versus User Perception." In: *International Journal of Information Security* 17.3 (2018).

[MBN14] Yan Michalevsky, Dan Boneh, and Gabi Nakibly. "Gyrophone: Recognizing Speech from Gyroscope Signals." In: *23rd USENIX Security Symposium*. 2014.

[Mic20] Microsoft Edge Team. *Mitigating Speculative Execution Side-Channel Attacks in Microsoft Edge and Internet Explorer*. (Visited on 07/25/2020). https://blogs.windows.com/msedgedev/2018/01/03/speculative-execution-mitigations-microsoft-edge-internet-explorer/.

[Mos20] *Most Android Users Running Outdated Security Patches: Report - CNET*. (Visited on 07/19/2020). https://www.cnet.com/news/most-android-users-running-outdated-security-patches-report-says/.

[Nar+16] Sashank Narain, Triet D. Vo-Huu, Kenneth Block, and Guevara Noubir. "Inferring User Routes and Locations Using Zero-Permission Mobile Sensors." In: *IEEE Symposium on Security and Privacy (S&P)*. 2016.

[Nas+18] Shoei Nashimoto, Daisuke Suzuki, Takeshi Sugawara, and Kazuo Sakiyama. "Sensor CON-Fusion: Defeating Kalman Filter in Signal Injection Attack." In: *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2018.

[Nin+18] Rui Ning, Cong Wang, Chunsheng Xin, Jiang Li, and Hongyi Wu. "DeepMag: Sniffing Mobile Apps in Magnetic Field through Deep Convolutional Neural Networks." In: *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 2018.

[Nov+15] Ed Novak, Yutao Tang, Zijiang Hao, Qun Li, and Yifan Zhang. "Physical Media Covert Channels on Smart Mobile Devices." In: *ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*. 2015.

[NSA20] *NSA TEMPEST Documents*. (Visited on 07/25/2020). http://cryptome.info/0001/nsa-tempest.htm.

[OO10] Keisuke Okamura and Yoshihiro Oyama. "Load-Based Covert Channels between Xen Virtual Machines." In: *International Symposium on Applied Computing (SAC)*. 2010.

[Opp99] Alan V. Oppenheim. *Discrete-Time Signal Processing*. Pearson Education India, 1999.

[Ore+15] Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, and Angelos D. Keromytis. "The Spy in the Sandbox: Practical Cache Attacks in JavaScript and Their Implications." In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2015.

[Ped+11] F. Pedregosa et al. "Scikit-Learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011).

[Per18] *Permissions Overview | Android Open Source Project*. (Visited on 11/22/2018). https://source.android.com/security/app-sandbox.

[Per+20] Mike Perry, Erinn Clark, Steven Murdoch, and Georg Koppen. *The Design and Implementation of the Tor Browser*. (Visited on 07/25/2020). https://2019.www.torproject.org/projects/torbrowser/design/.

[PPG13] Said Pertuz, Domenec Puig, and Miguel Ángel García. "Analysis of Focus Measure Operators for Shape-from-Focus." In: *Pattern Recognition* 46.5 (2013).

[Pim+14] Marco A. F. Pimentel, David A. Clifton, Lei A. Clifton, and Lionel Tarassenko. "A Review of Novelty Detection." In: *Signal Processing* 99 (2014).

[QS01] Jean-Jacques Quisquater and David Samyde. "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards." In: *International Conference on Research in Smart Cards (E-smart)*. 2001.

[RKM17] Edith Ramirez, Maureen K. Ohlhausen, and Terrell McSweeny. *Cross-Device Tracking: An FTC Staff Report*. Technical report. Federal Trade Comission, 2017.

[Rao09] Nannapaneni Narayana Rao. *Fundamentals of Electromagnetics for Electrical and Computer Engineering*. Prentice Hall, 2009.

[RC99] Irving S. Reed and Xuemin Chen. "BCH Codes." In: *Error-Control Coding for Data Networks*. Springer, 1999.

[Ros15] Adrian Rosebrock. *Blur Detection with OpenCV*. PyImageSearch. (Visited on 02/07/2022). https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/.

[RRR16] Michael Rushanan, David Russell, and Aviel D. Rubin. "MalloryWorker: Stealthy Computation and Covert Channels Using Web Workers." In: *International Workshop on Security and Trust Management (STM)*. 2016.

[Sch+11] Roman Schlegel, Kehuan Zhang, Xiaoyong Zhou, Mehool Intwala, Apu Kapadia, and XiaoFeng Wang. "Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones." In: *Annual Network and Distributed System Security Symposium (NDSS)*. 2011.

[Sch+18] Matthias Schulz, Jakob Link, Francesco Gringoli, and Matthias Hollick. "Shadow Wi-Fi: Teaching Smartphones to Transmit Raw Signals and to Extract Channel State Information to Implement Practical Covert Channels over Wi-Fi." In: *16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2018.

[Sch+17] Michael Schwarz, Clémentine Maurice, Daniel Gruss, and Stefan Mangard. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript." In: *21th International Conference on Financial Cryptography and Data Security (FC)*. 2017.

[Sen18] *Sensors Overview | Android Developers*. (Visited on 10/23/2018). https://www.techspot.com/news/66899-mobile-web-browsing-becomes-more-popular-than-desktop.html.

[She+21] Cheng Shen, Tian Liu, Jun Huang, and Rui Tan. "When LoRa Meets EMR: Electromagnetic Covert Channels Can Be Super Resilient." In: *IEEE Symposium on Security and Privacy (S&P)*. 2021.

[Shu+19] Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. "Robust Website Fingerprinting through the Cache Occupancy Channel." In: *28th USENIX Security Symposium*. 2019.

[SXA16] Laurent Simon, Wenduan Xu, and Ross Anderson. "Don't Interrupt Me While I Type: Inferring Text Entered through Gesture Typing on Android Keyboards." In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2016.3 (2016).

[Sol+10] Ashkan Soltani, Shannon Canty, Quentin Mayo, Lauren Thomas, and Chris Jay Hoofnagle. "Flash Cookies and Privacy." In: *Intelligent Information Privacy Management. Papers from the 2010 AAAI Spring Symposium.* 2010.

[Son+15] Yunmok Son, Hocheol Shin, Dongkwan Kim, Young-Seok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, Yongdae Kim, et al. "Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors." In: *24th USENIX Security Symposium*. 2015.

[Spr+16] Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. "Exploiting Data-Usage Statistics for Website Fingerprinting Attacks on Android." In: *9th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. 2016.

[Spr+18] Raphael Spreitzer, Felix Kirchengast, Daniel Gruss, and Stefan Mangard. "ProcHarvester: Fully Automated Analysis of Procfs Side-Channel Leaks on Android." In: *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2018.

[SPM18] Raphael Spreitzer, Gerald Palfinger, and Stefan Mangard. "SCAnDroid: Automated Side-Channel Analysis of Android APIs." In: *11th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec)*. 2018.

[Str21] *Strategy Analytics: Half the World Owns a Smartphone*. (Visited on 02/04/2022). https://news.strategyanalytics.com/press-releases/press-release-details/2021/Strategy-Analytics-Half-the-World-Owns-a-Smartphone/default.aspx.

[Retr18] Lexi Sydow and Sam Cheney. *2017 Retrospective: A Monumental Year for the App Economy*. (Visited on 07/17/2018). https://www.appannie.com/en/insights/market-data/app-annie-2017-retrospective/.

[Sze19] Jakub Szefer. "Survey of Microarchitectural Side and Covert Channels, Attacks, and Defenses." In: *Journal of Hardware and Systems Security* 3.3 (2019).

[Tal12] Michael Talbot-Smith. *Audio Engineer's Reference Book*. Focal Press, 2012.

[Tha+20] Kevin Sam Tharayil, Benyamin Farshteindiker, Shaked Eyal, Nir Hasidim, Roy Hershkovitz, Shani Houri, Ilia Yoffe, Michal Oren, and Yossi Oren. "Sensor Defense In-Software (SDI): Practical Software Based Detection of Spoofing Attacks on Position Sensors." In: *Engineering Applications of Artificial Intelligence* 95 (2020).

[TS19] Shanquan Tian and Jakub Szefer. "Temporal Thermal Covert Channels in Cloud FPGAs." In: *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2019.

[Tor20] *Tor Bug Tracker & Wiki. 16110: Improve Time Resolution Defense*. (Visited on 07/25/2020). https://trac.torproject.org/projects/tor/ticket/16110.

[Tri+17] Timothy Trippel, Ofir Weisse, Wenyuan Xu, Peter Honeyman, and Kevin Fu. "WALNUT: Waging Doubt on the Integrity of Mems Accelerometers with Acoustic Injection Attacks." In: *IEEE European Symposium on Security and Privacy (EuroS&P)*. 2017.

[Tru22] *TrueTime for Android*. (Visited on 02/07/2022). https://github.com/instacart/truetime-android.

[ULM15] Randika Upathilake, Yingkun Li, and Ashraf Matrawy. "A Classification of Web Browser Fingerprinting Techniques." In: *7th International Conference on New Technologies, Mobility and Security (NTMS)*. 2015.

[Use20] *User-Agent - HTTP | MDN*. (Visited on 07/22/2020). https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent.

[vEck85] Wim van Eck. "Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk?" In: *Computers & Security* 4.4 (1985).

[Vis20] *Visual Studio App Center | Visual Studio*. (Visited on 07/22/2020). https://visualstudio.microsoft.com/app-center/.

[Wag20] Luke Wagner. *Mitigations Landing for New Class of Timing Attack - Mozilla Security Blog*. (Visited on 07/25/2020). https://blog.mozilla.org/security/2018/01/03/mitigations-landing-new-class-timing-attack/.

[WPS18] Rick Waldron, Mikhail Pozdnyakov, and Alexander Shalamov. *Generic Sensor API. W3C Candidate Recommendation*. (Visited on 10/23/2018). https://www.techspot.com/news/66899-mobile-web-browsing-becomes-more-popular-than-desktop.html.

[WG13] Tao Wang and Ian Goldberg. "Improved Website Fingerprinting on Tor." In: *12th Annual ACM Workshop on Privacy in the Electronic Society (WPES)*. 2013.

[Wan+17] Zhengbo Wang, Kang Wang, Bo Yang, Shangyuan Li, and Aimin Pan. *Sonic Gun to Smart Devices*. Alibaba Security, 2017.

[WM90] Toshio Watanabe and Henrik Moller. "Hearing Thresholds and Equal Loudness Contours in Free Field at Frequencies below 1 kHz." In: *Journal of Low Frequency Noise, Vibration and Active Control* 9.4 (1990).

[Web18] *WebKit WebView | Android Developers*. (Visited on 11/28/2018). https://developer.android.com/studio/command-line/adb.

[Whi20] Ethan White. *CPU Correlation Attacks*. (Visited on 07/25/2020). https://www.ethanlw.me/cpu-correlation.

[WCR72] L.S. Whittle, S. J. Collins, and D. W. Robinson. "The Audibility of Low-Frequency Sounds." In: *Journal of Sound and Vibration* 21.4 (1972).

[WG93] Brett Wilson and Zabih Ghassemlooy. "Pulse Time Modulation Techniques for Optical Communications: A Review." In: *IEE Proceedings J (Optoelectronics)* 140.6 (1993).

[Yan+20] Chen Yan, Hocheol Shin, Connor Bolton, Wenyuan Xu, Yongdae Kim, and Kevin Fu. "SoK: A Minimalist Approach to Formalizing Analog Sensor Security." In: *IEEE Symposium on Security and Privacy (S&P)*. 2020.

[Yan+15] Lin Yan, Yao Guo, Xiangqun Chen, and Hong Mei. "A Study on Power Side Channels on Mobile Devices." In: *7th Asia-Pacific Symposium on Internetware* (2015).

[Yan+17] Qing Yang, Paolo Gasti, Gang Zhou, Aydin Farajidavar, and Kiran S. Balagani. "On Inferring Browsing Activity on Smartphones via USB Power Analysis Side-Channel." In: *IEEE Transactions on Information Forensics and Security* 12.5 (2017).

[ZP14] Alenka Zajić and Milos Prvulovic. "Experimental Demonstration of Electromagnetic Information Leakage from Modern Processor-Memory Systems." In: *IEEE Transactions on Electromagnetic Compatibility* 56.4 (2014).

[ZAB07] Sebastian Zander, Grenville J. Armitage, and Philip Branch. "A Survey of Covert Channels and Countermeasures in Computer Network Protocols." In: *IEEE Communications Surveys & Tutorials* 9.1-4 (2007).

[ZZK20] Zihao Zhan, Zhenkai Zhang, and Xenofon D. Koutsoukos. "BitJabber: The World's Fastest Electromagnetic Covert Channel." In: *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2020.

[Zha+17] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. "DolphinAttack: Inaudible Voice Commands." In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2017.

[ZW09] Kehuan Zhang and XiaoFeng Wang. "Peeping Tom in the Neighborhood: Keystroke Eavesdropping on Multi-User Systems." In: *18th USENIX Security Symposium*. 2009.

[Zha+18] Xiaokuan Zhang, Xueqiang Wang, Xiaolong Bai, Yinqian Zhang, and XiaoFeng Wang. "OS-level Side Channels without Procfs: Exploring Cross-App Information Leakage on iOS." In: *Annual Network and Distributed System Security Symposium (NDSS)*. 2018.

[ZNL19] Yue Zhao, Zain Nasrullah, and Zheng Li. "PyOD: A Python Toolbox for Scalable Outlier Detection." In: *Journal of Machine Learning Research* 20.96 (2019).

[Zho+13] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A. Gunter, and Klara Nahrstedt. "Identity, Location, Disease and More: Inferring Your Secrets from Android Public Resources." In: *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2013.