SoK: Systematic Classification of Side-Channel Attacks on Mobile Devices

Raphael Spreitzer*, Veelasha Moonsamy†, Thomas Korak* and Stefan Mangard*
*Graz University of Technology, IAIK, Graz, Austria
†Radboud University, Digital Security Group, Nijmegen, The Netherlands

***Abstract*—Side-channel attacks on mobile devices have gained increasing attention since their introduction in 2007. While traditional side-channel attacks, such as power analysis attacks and electromagnetic analysis attacks, required physical presence of the attacker as well as expensive equipment, an (unprivileged) application is all it takes to exploit the leaking information on modern mobile devices. Given the vast amount of sensitive information that are stored on smartphones, the ramifications of side-channel attacks affect both the security and privacy of users and their devices.**

In this paper, we propose a new categorization system for side-channel attacks on mobile devices, which is necessary since side-channel attacks have evolved significantly since their introduction during the smartcard era. Our proposed classification system allows to analyze side-channel attacks systematically, and facilitates the development of novel countermeasures. Besides this new categorization system, the extensive overview of existing attacks and attack strategies provides valuable insights on the evolving field of side-channel attacks on mobile devices. We conclude by discussing open issues and challenges in this context and outline possible future research directions.

1. Introduction

Side-channel attacks exploit (unintended) information leakage of computing devices or implementations to infer sensitive information. Starting with the seminal works of Kocher [1], Kocher et al. [2], Quisquater and Samyde [3], as well as Mangard et al. [4], many follow-up papers considered attacks against cryptographic implementations to exfiltrate key material from smartcards by means of timing information, power consumption, or electromagnetic (EM) emanation. Although these "traditional" side-channel attacks required the attacker to be in physical possession of the device, different attacks assumed different types of attackers and different levels of invasiveness. To systematically analyze side-channel attacks, they have been categorized along the following two orthogonal axes:

1) *Active* vs *passive*: Depending on whether the attacker actively influences the behavior of the device or only passively observes leaking information.
2) *Invasive* vs *semi-invasive* vs *non-invasive*: Depending on whether or not the attacker removes the passivation layer of the chip, depackages the chip, or does not manipulate the packaging at all.

However, with the era of cloud computing, the scope and the scale of side-channel attacks have changed significantly in the early 2000s. While early attacks required attackers to be in physical possession of the device, newer side-channel attacks, for example, cache-timing attacks [5]–[8] or DRAM row buffer attacks [9], are conducted remotely by executing malicious software in the targeted cloud environment. In fact, the majority of recently published side-channel attacks rely on passive attackers and are strictly non-invasive.

With the advent of mobile devices, and in particular the plethora of embedded features and sensors, even more sophisticated side-channel attacks targeting smartphones have been proposed since around the year 2010. For example, attacks allow to infer keyboard input on touchscreens via sensor readings from native apps [10]–[12] and websites [13], to deduce a user's location via the power consumption available from the proc filesystem (procfs) [14], and to infer a user's identity, location, and diseases [15] via the procfs.

Although side-channel attacks and platform security are already well-studied topics, it must be noted that smartphone security and associated privacy aspects differ from platform security in the context of smartcards, desktop computers, and cloud computing. Especially the following *key enablers* allow for more devastating attacks on mobile devices.

1) *Always-on and portability*: First and foremost, mobile devices are always turned on and due to their mobility they are carried around at all times. Thus, they are tightly integrated into our everyday life.
2) *Bring your own device (BYOD)*: To decrease the number of devices carried around, employees are encouraged to use private devices to process corporate data and to access corporate infrastructure, which clearly indicates the importance of secure mobile devices.
3) *Ease of software installation*: Due to the appification [16] of mobile devices, *i.e.*, where there is an app for almost everything, additional software can be installed easily by means of established app markets. Hence, malware can also be spread at a fast pace.
4) *OS based on Linux kernel*: Modern mobile operating systems (OS), for example, Android, are based on the Linux kernel. The Linux kernel, however, has initially been designed for desktop machines and information or features that are considered harmless on these platforms turn out to be an immense security and/or privacy threat on mobile devices (cf. [17]).
5) *Features and sensors*: Last but not least, these devices include many features and sensors, which are not

present on traditional platforms. Due to the inherent nature of mobile devices (always-on and carried around, inherent input methods, etc.), such features often allow for devastating side-channel attacks [10]–[15]. Besides, these sensors have also been used to attack external hardware, such as keyboards [18], [19], and computer hard drives [20], to infer videos played on TVs [21], and even to attack 3D printers [22], [23], which clearly demonstrates the immense power of mobile devices.

Today's smartphones are vulnerable to (all or most of the) existing side-channel attacks against smartcards and cloud computing infrastructures. However, due to the above mentioned *key enablers*, a new area of side-channel attacks has evolved. The appification [16] of mobile platforms—*i.e.*, where there is an app for anything—allows to easily target devices and users at an unprecedented scale compared to the smartcard and the cloud setting. Yet again, the majority of these attacks are passive and non-invasive, which means that the existing side-channel classification system is not appropriate anymore as it is too coarse grained for a systematic categorization of modern side-channel attacks against mobile devices.

In this work, we aim to close this gap by establishing a new categorization system for modern side-channel attacks on mobile devices. We take a step back and discuss existing side-channel attacks in a broader context. The resulting survey of existing literature aims to provide a better understanding of threats arising from modern side-channel attacks. Then, we systematically categorize existing side-channel attacks on smartphones and critically discuss resulting security and privacy implications. Ultimately, we identify commonalities of side-channel attacks and thereby provide insights to the immense number of different attack possibilities. Overall, the goal is that the identified commonalities of such attacks allow researchers to cope with these attacks on a larger scale. Although dedicated countermeasures to prevent specific attacks have already been proposed, their effectiveness has not been investigated extensively. In this paper, we aim to foster further research through the development and evaluation of effective countermeasures. Moreover, the mobile ecosystem makes it hard to distribute security patches (*i.e.*, possible countermeasures) and, thus, OS designers also struggle with these vulnerabilities. We aim to address these issues and challenges by introducing a new categorization system that allows for a more systematic investigation of modern side-channel attacks and possible mitigation techniques.

1.1. High-Level Categorization

It is important to note that side-channel attacks against smartphones can be launched by attackers who are in physical possession of these devices and also by remote attackers who managed to spread a seemingly innocuous application via any of the existing app stores. In some cases such side-channel attacks can even be launched via a website and, thus, without relying on the user to install an app. Nevertheless, in today's appified software platforms where apps are distributed easily via available app markets, an attack scenario requiring the user to install a malicious game is entirely practical. Interestingly, side-channel attacks on smartphones allow for exploitation of physical properties as well as software properties. More specifically, a malicious application can exploit the accelerometer sensor in order to attack the user input (cf. [10], [11]), which is due to the inherent input method relying on touchscreens. In addition, attacks can also be conducted by exploiting software features provided by the Android API or the mobile OS itself (cf. [14], [15]). This clearly indicates that smartphones significantly broaden the scope as well as the scale of attacks, which is also stressed by the numerous scientific contributions published in this area of research.

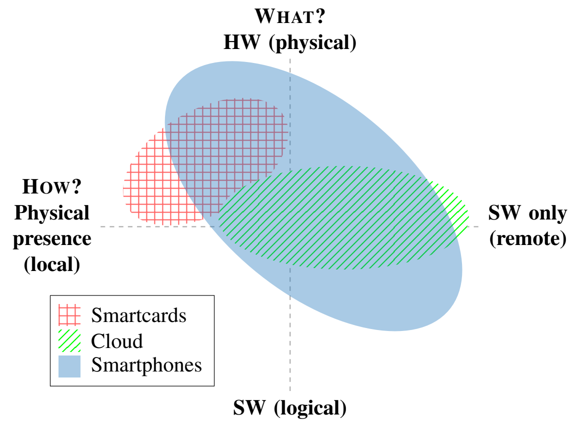

Figure 1. Scope of attacks for smartcards, cloud infrastructures, and smartphones.

Figure 1 illustrates a high-level categorization system for side-channel attacks and how *existing* side-channel attacks against smartcards, cloud computing infrastructures, and smartphones relate to it. We indicate the type of information that is exploited (WHAT?) and how the adversary learns the leaking information (HOW?) on the y-axis and x-axis, respectively. For instance, attackers usually exploit hardware-based information leakage (or physical properties) [4] of smartcards by measuring, for example, the power consumption with an oscilloscope. In order to exploit this information, the attacker must be in possession of the device under attack. In contrast, side-channel attacks against cloud-computing infrastructures do not (necessarily) require the attacker to be physically present—unless we consider a malicious cloud provider—as the attacker is able to remotely execute software. Usually, these attacks exploit microarchitectural behavior (like cache attacks [5]–[8]) or software features in order to infer secret information from co-located processes. Even more manifold and diverse side-channel attacks have been proposed for smartphones, which is indicated by the larger area in Figure 1. These manifold side-channel attacks mainly result from the five aforementioned *key-enablers*. In the remainder of this paper we will refine this high-level categorization system in order to systematically analyze modern side-channel attacks.

1.2. Outline

Section 2 introduces the basic notion of side-channel attacks, discusses different types of information leaks, and provides a definition for software-only side-channel attacks. In Section 3, we introduce our new categorization system for modern side-channel attacks. We survey existing attacks in Sections 4, and 5, and we discuss existing countermeasures in Section 6. We classify existing attacks according to our newly introduced classification system in Section 7. Finally, we discuss open issues, challenges, and future research directions in Section 8 and conclude in Section 9.

2. Taxonomy

In this section, we define the general notion of side-channel attacks and we establish the boundaries between side-channel attacks and other attacks on mobile devices. We stress that side-channel attacks do not exploit specific software vulnerabilities of the OS or any library, but instead exploit available information that is (in some cases) published for benign reasons. Furthermore, we also discuss how the key enablers presented above allow for so-called *software-only attacks* on today's smartphones.

2.1. Basic Concept of Side-Channel Attacks

Passive Side-Channel Attacks. The general notion of a passive side-channel attack can be described by means of three main components, *i.e.*, *target*, *side-channel vector*, and *attacker*. A *target* represents anything of interest to possible attackers. During the computation or operation of the target, it influences a *side-channel vector* and thereby emits potential sensitive information. An *attacker* who is able to observe these side-channel vectors potentially learns useful information related to the actual computations or operations performed by the target.

Active Side-Channel Attacks. In addition to passively observing leaking information, an active attacker also tries to tamper with the device or to modify/influence the targeted device via a side-channel vector, e.g., via an external interface or environmental conditions. Thereby, the attacker aims to influence the computation/operation performed by the device in a way that leads to malfunctioning, which in turn allows for possible attacks either indirectly via the leaking side-channel information or directly via the (erroneous) output of the targeted device.

Figure 2 illustrates the general notion of side-channel attacks. A target emits specific side-channel information as it influences specific side-channel vectors. For example, physically operating a smartphone via the touchscreen, *i.e.*, the touchscreen input represents the target, causes the smartphone to undergo specific movements and accelerations in all three dimensions. In this case, one possible side-channel vector is the acceleration of the device, which can be observed via the embedded accelerometer sensor and accessed by an app via the official Sensor API. The relations defined via the solid arrows, *i.e.*, *target* $\rightarrow$ *side-channel vector* $\rightarrow$ *attacker*, represent passive side-channel attacks. The relations defined via the dashed arrows, *i.e.*, *target* $\dashleftarrow$ *side-channel vector* $\dashleftarrow$ *attacker*, represent active side-channel attacks where the attacker—in addition to passively observing leaking information—also tries to actively influence/manipulate the target via a side-channel vector.

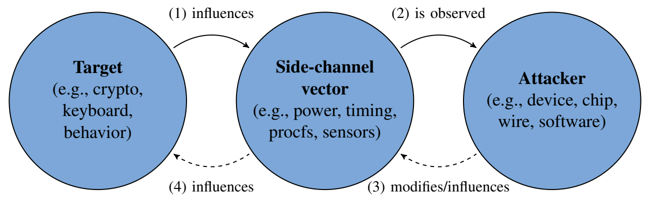

Figure 2. General notion of active and passive side-channel attacks. A passive side-channel attack consists of steps (1) and (2), whereas an active side-channel attack also includes steps (3) and (4).

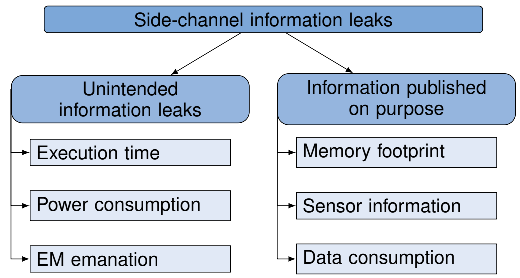

Figure 3. Categorization of side-channel information leaks.

Irrespective of whether an attacker is passive or active, we consider only side-channel attacks. These attacks *do not* exploit software bugs or anomalies within the OS or apps like, e.g., buffer overflow attacks. While software bugs or anomalies are in general easy to fix, side-channel information leaks are not that trivial to detect and fix.

2.2. Types of Side-Channel Information Leaks

Considering existing side-channel attacks on mobile devices, we identify two categories of side-channel information leaks, namely *unintended information leaks* and *information published on purpose*. Figure 3 depicts these two types of information leaks. Informally, side-channel attacks exploiting unintended information leaks of computing devices can also be considered as "traditional" side-channel attacks since this category has already been extensively analyzed and exploited during the smartcard era [4]. For example, unintended information leaks include the execution time, the power consumption, or the electromagnetic emanation of a computing device. This type of information leak is considered as unintended because smartcard designers and developers did not plan to leak the timing information or power consumption of computing devices on purpose.

The second category of information leaks (referred to as *information published on purpose*) is mainly a result of the ever-increasing number of features provided by today's smartphones. In contrast to unintended information leaks, the exploited information is published on purpose and for benign reasons. For instance, specific features require the device to share (seemingly harmless) information and resources with all applications running in parallel on the system. This information as well as specific resources are either shared by the OS directly (via the procfs) or through the official Android API.[1] Although this information is extensively used by many legitimate applications for benign purposes[2], it sometimes turns out to leak sensitive information and, thus, leads to devastating side-channel attacks. The fundamental design weakness of assuming information as being innocuous in the first place also means that it is not protected by dedicated permissions. Many investigations have impressively demonstrated that such seemingly harmless information can be used to infer sensitive information that is otherwise protected by dedicated security mechanisms, such as permissions. Examples include the memory footprint [25] and the data-usage statistics [26] that have been shown to leak a user's browsing behavior and, hence, bypass the `READ_HISTORY_BOOKMARKS` permission.

Furthermore, the second category seems to be more dangerous in the context of smartphones as new features are added frequently and new software interfaces allow to access an unlimited number of unprotected resources. Even developers taking care of secure implementations in the sense of unintended information leaks, e.g., by providing constant-time crypto implementations, and taking care of possible software vulnerabilities like buffer overflow attacks, inevitably leak sensitive information due to shared resources, the OS, or the Android API. Additionally, the provided software interfaces to access information and shared resources allow for so-called *software-only attacks*, *i.e.*, side-channel attacks that only require the execution of malicious software. This clearly represents an immense threat as these attacks (1) do not exploit any obvious software vulnerabilities, (2) do not rely on specific privileges or permissions, and (3) can be conducted remotely via malicious apps or even websites.

2.3. Software-only Side-Channel Attacks

As previously mentioned, side-channel attacks either exploit physical properties or logical properties (software features). Irrespective of whether a physical property (e.g., execution time [6] and power consumption [14]) or a software feature (e.g., memory footprint available via the procfs [25] and data-usage statistics [15], [26]) are exploited, smartphones allow many of these side-channel information leaks to be exploited by means of *software-only attacks*. More specifically, software-only attacks allow to exploit leaking information without additional equipment that was required for traditional side-channel attacks. For example, an oscilloscope is necessary to measure the power consumption of a smartcard during its execution, or an EM probe is necessary to measure the EM emanation. In contrast, today's smartphones allow an impressive number of side-channel leaks to be exploited via software-only attacks. Besides, an attack scenario that requires the user to install an (unprivileged) application—*i.e.*, an addictive game—is entirely reasonable in an appified ecosystem.

For the generic class of side-channel attacks, it does not matter whether the leaking information is collected via dedicated equipment or whether an unprivileged app collects the leaking information directly on the device under attack (software-only attacks). Interestingly, however, the immense amount of information published on purpose allows to observe physical properties of the device as well as physical interactions with the device. Consequently, *software-only side channel attacks* have gained increasing attention in the last few years and impressive attacks are continuously published by the scientific community.

Runtime-Information Gathering Attacks. Zhang et al. [17] coined the term runtime-information gathering (RIG) attack, which refers to attacks that require a malicious app to run side-by-side with a victim app on the same device in order to collect runtime information of the victim. This generic class of attacks also includes a subset of side-channel attacks, especially side-channel attacks that can be launched via software-only attacks. However, RIG attacks also include attacks that we do not consider as side-channel attacks. For example, RIG attacks also include attacks where apps request permissions which are exploited for (more obvious) attacks, e.g., requesting the `RECORD_AUDIO` permission in order to eavesdrop on phone conversations.

Screenmilker [27]—an attack exploiting ADB capabilities to take screenshots programmatically—is also considered being a RIG attack. We do not consider such attacks as side-channel attacks because these attacks exploit implementation flaws, *i.e.*, the exploited screenshot tool does not implement any authentication mechanism and hence any application can take screenshots programmatically. Similarly, we do not consider buffer overflow attacks as a means to launch active side-channel attacks because buffer overflow attacks represent a software vulnerability. Side-channel attacks, however, attack targets that are secure from a software perspective and still leak information unintentionally. Furthermore, software vulnerabilities, e.g., missing authentication mechanisms and buffer overflow attacks, can be fixed easily, whereas side-channel attacks usually cannot be fixed or prevented that easily.

Figure 4 illustrates the new type of software-only side-channel attacks that allow to exploit both, physical properties as well as software features (logical properties), without additional equipment. Hence, software-only attacks allow for large-scale attacks against an immense number of smartphone users at the same time. As software-only attacks also rely on software being executed side-by-side with the victim application, software-only attacks are a sub-category of RIG attacks. It should be noted that physical attacks on smartphones might still rely on dedicated hardware and also some logical attacks can also be conducted without running software on the device under attack. Such attacks

1. In the literature some of the information leaks through the procfs are also denoted as *storage side channels* [24].

2. For example, the data-usage statistics, *i.e.*, the amount of incoming and outgoing network traffic, is publicly available for all applications.

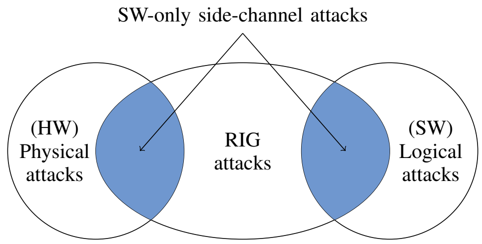

Figure 4. Relations between different types of attacks. SW-only side-channel attacks allow to exploit physical as well as logical properties.

are covered by the non-overlapping areas of "(HW) physical attacks" and "(SW) logical attacks" in Figure 4. However, physical attacks that cannot be conducted by running software on the targeted device are more targeted attacks as they require attackers to be in physical presence of the device.

2.4. Adversary Model and Attack Scenario

In contrast to traditional attacks that require the attacker to have the device under physical control or to be physically present with the targeted victim, the adversary model for most (existing) side-channel attacks on smartphones shifted the scope to remote software execution by means of apps or websites. This also dramatically increases the scale of these attacks. While traditional side-channel attacks targeted only a few devices, modern side-channel attacks target possibly millions of devices or users at the same time. With this general overview of the adversary model in mind, most software-only attacks usually consider the following two-phase attack scenario:

Training phase: In the training phase, the attacker "profiles" specific actions or events of interest, either during an online phase on the attacked device or during an offline phase in dedicated environments. Sometimes this training phase includes the establishment of a machine-learning model, e.g., a supervised classifier. More abstractly, the attacker builds specific "templates" based on dedicated events of interest. In addition, the attacker crafts an app (or website) that ideally does not require any permissions or privileges in order to avoid raising the user's suspicion. This app is used in the attack phase to gather leaking information.

Attack phase: The attack phase usually consists of three steps. (1) A malicious application—that is hidden inside a popular app—is spread via existing app markets. After installation, this malicious app waits in the background until the targeted app/action/event starts and then (2) it observes the leaking side-channel information. Based on the gathered information, (3) it employs the previously established model or templates to infer secret information. Depending on the complexity of the inference mechanism, e.g., the complexity of the machine-learning classifier, the gathered side-channel information could alternatively be sent to a remote server, which then performs the heavy computations to infer the secret information.

3. A New Categorization System

Our new categorization system, depicted in Figure 5, classifies side-channel attacks along three axes:

1) *Passive* vs *active*: This category distinguishes between attackers that passively observe leaking side-channel information and attackers that also actively influence the target via any side-channel vector. For instance, an attacker can manipulate the target, its input, or its environment via any side-channel vector in order to subsequently observe leaking information via abnormal behavior of the target (cf. [4]). However, for both cases, we always assume a correct implementation without any obvious software vulnerabilities like, e.g., buffer overflow vulnerabilities.
2) *Physical properties* vs *logical properties*: This category classifies side-channel attacks according to the exploited information, *i.e.*, depending on whether the attack exploits physical properties (hardware) or logical properties (software features). Some attacks that exploit software features also refer to such attacks as storage side-channels (cf. [24]). However, not all attacks exploiting software features target information that is actually stored somewhere.
3) *Local attackers* vs *vicinity attackers* vs *remote attackers*: Side-channel attacks are classified depending on whether or not the attacker must be in physical proximity/vicinity of the target. *Local attackers* clearly must be in (temporary) possession of the device or at least in close proximity. *Vicinity attackers* are able to wiretap or eavesdrop the network communication of the target or to be somewhere in the vicinity of the target. *Remote attackers* only rely on software execution on the targeted device. Clearly, the scale increases significantly for these three attackers as a local attacker relies on stronger assumptions than a remote attacker. Especially the immense number of software-only attacks (that allow to conduct side-channel attacks remotely) stress the need for this category.

Figure 5 illustrates our new categorization system. We distinguish between *active* and *passive* attackers along the (right) y-axis. Passive attacks are classified above the x-axis and active attacks are classified below the x-axis. The (left) y-axis distinguishes the exploited side-channel vector, *i.e.*, *physical properties* and *logical properties*. As both of these categories can be exploited by passive as well as active attackers, these two categories are mirrored along the x-axis. The x-axis categorizes side-channel attacks according to the attacker's proximity to the targeted device. For instance, some attacks require an attacker to have access to the targeted device or even to have access to components within the device, e.g., the attacker might remove the back cover in order to measure the EM emanation of the chip. Stronger adversaries (with weaker assumptions) might rely on wiretapping techniques. The strongest adversaries only rely on unprivileged applications being executed on the targeted device or even only that the victim browses a malicious website.

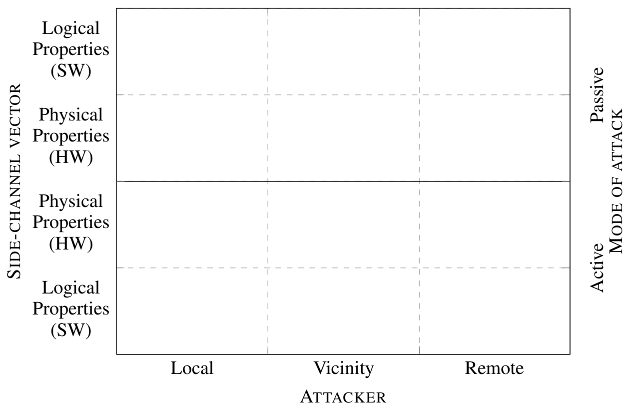

Figure 5. Overview of our proposed classification system for side-channel attacks: (1) active vs passive, (2) logical properties vs physical properties, (3) local attackers vs vicinity attackers vs remote attackers.

Subsequently, we briefly survey existing attacks according to our new classification system. We start with passive attacks and classify these attacks along the attacker's vicinity to the target in Section 4. In addition to passive attacks we also survey active attacks in Section 5. Hence, the following sections provide an extensive survey of existing attacks.

4. Passive Attacks

Passive attacks only observe leaking information without actively influencing or manipulating the target.

4.1. Local Side-Channel Attacks

Below we survey side-channel attacks that require a local adversary. We start with traditional side-channel attacks that aim to break insecure cryptographic implementations (of mathematically secure primitives). Besides, we discuss attacks that target the user's interaction with the device as well as the user's input on the touchscreen, *i.e.*, attacks that result from the inherent nature of mobile devices.

Power Analysis Attacks. The actual power consumption of a device or implementation depends on the processed data and executed instructions. Power analysis attacks exploit this information leak to infer sensitive information.

<u>Attacks</u>. Traditional side-channel attacks exploiting the power consumption of smartcards [4] have also been applied on mobile devices. For instance, attacks targeting symmetric cryptographic primitives [28] as well as asymmetric primitives [29]–[31] have been successfully demonstrated. Such attacks have even been conducted with low-cost equipment, as has been impressively demonstrated by Genkin et al. [29]. Furthermore, the power consumption of smartphones allows to identify running applications [32].

Electromagnetic Analysis Attacks. Another way to attack the leaking power consumption of computing devices is to exploit electromagnetic emanations.

<u>Attacks</u>. Gebotys et al. [33] demonstrated attacks on software implementations of AES and ECC on Java-based PDAs. Later on, Nakano et al. [34] attacked ECC and RSA implementations of the default crypto provider (JCE) on Android smartphones and Belgarric et al. [31] attacked the ECDSA implementation of Android's Bouncy Castle.

Smudge Attacks. The most common input method on mobile devices is the touchscreen, *i.e.*, users tap and swipe on the screen with their fingers. However, due to the inherent nature of touchscreens, users always leave residues in the form of fingerprints and smudges on the screen.

<u>Attacks</u>. Aviv et al. [35] pointed out that side-channel attacks can be launched due to specific interactions with the smartphone or touchscreen-based devices in general. More specifically, forensic investigations of smudges (oily residues from the user's fingers) on the touchscreen allow to infer unlock patterns. Even after cleaning the phone or placing the phone into the pocket, smudges seem to remain most of the time. Hence, smudges are quite persistent which increases the threat of smudge attacks. Follow-up work considering an attacker who employs fingerprint powder to infer keypad inputs has been presented by Zhang et al. [36] and also an investigation of the heat traces left on the screen by means of thermal cameras has been performed [37].

Shoulder Surfing and Reflections. Touchscreens of mobile devices optically/visually emanate the displayed content. Often these visual emanations are reflected by objects in the environment, e.g., sunglasses, tea pots, etc. [38], [39].

<u>Attacks</u>. Maggi et al. [40] observed that touchscreen input can be recovered by observing the visual feedback (pop-up characters) on soft keyboards during the user input. Raguram et al. [41], [42] observed that reflections, e.g., on the user's sunglasses, can also be used to recover input typed on touchscreens. However, the attacker needs to point the camera, used to capture the reflections, directly on the targeted user. Subsequently, they rely on computer vision techniques and machine learning techniques to infer the user input from the captured video stream. Xu et al. [43] extended the range of reflection-based attacks by considering reflections of reflections. Although, they do not rely on the visual feedback of the soft keyboard but instead track the user's fingers on the smartphone while interacting with the device.

Hand/Device Movements. Many input methods on various devices rely on the user operating the device with her hands and fingers. For instance, users tend to hold the device in their hands while operating it with their fingers.

<u>Attacks</u>. Similar to reflections, Shukla et al. [44] proposed to monitor hand movements as well as finger movements—without directly pointing the camera at the targeted screen—in order to infer entered PIN inputs. Similarly, Sun et al. [45] monitored the backside of tablets during user input and detected subtle motions that can be used to infer keystrokes. Yue et al. [46] proposed an attack where the input on touch-enabled devices can be estimated from a video of a victim tapping on a touch screen.

4.2. Vicinity Side-Channel Attacks

In this section, we briefly recall attacks that require the attacker to be in the vicinity of the targeted user/device. For example, the attacker compromises any infrastructure facility within the user's environment.

Network Traffic Analysis. In general, the encryption of messages transmitted between two parties only hides the actual content, while specific meta data like, e.g., the overall amount of data, is not protected. This observation can be used to infer sensitive information about the transmitted content and about the communicating parties.

Attacks. Network traffic analysis has been extensively studied in the context of website fingerprinting attacks. These attacks [47]–[51] aim to infer visited websites by wiretapping network connections in order to observe traffic signatures, e.g., unique packet lengths, inter-packet timings, etc., and even work in case the traffic is routed through Tor.

While most website fingerprinting attacks target the network communication in general, attacks explicitly targeting mobile devices also exist. For instance, Stöber et al. [52] assumed that an adversary can eavesdrop on the UMTS transmission and showed that smartphones can be fingerprinted according to the background traffic generated by installed apps. Conti et al. [53] consider an adversary who controls WiFi access points near the targeted device and, thereby, infer specific app actions like sending mails, posting Facebook status messages, etc. In a similar setting, many papers demonstrated the feasibility to fingerprint specific apps and actions performed in specific apps based on traffic analysis techniques (cf. [54]–[59]).

While the above presented attacks exploit logical properties, *i.e.*, the fact that encrypted packets do not hide meta data, a recent work by Schulz et al. [60] showed that also hardware properties can be exploited. Therefore, they exploit the electromagnetic emanation of Ethernet cables to eavesdrop on transmitted packets on the wire.

User Identification. The identification (or localization) of specific users within specific areas or environments is considered a privacy risk.

Attacks. Conti et al. [61] demonstrated that wall-socket smart meters that capture the power consumption of plugged devices can be used to identify specific users/notebooks. Although Conti et al. demonstrated the power of their attack by using notebooks, it is likely that the same attack works for smartphones as well.

WiFi Signal Monitoring. WiFi devices continuously monitor the wireless channel (channel state information (CSI)) to effectively transmit data. This is necessary as environmental changes cause the CSI values to change.

Attacks. Ali et al. [62] observed that even finger motions impact wireless signals and cause unique patterns in the time-series of CSI values. In a setting with a sender (notebook) and a receiver (WiFi router), they showed that keystrokes on an external keyboard cause distortions in the WiFi signal. By monitoring how the CSI values change, they are able to infer the entered keys. Later on, Zhang et al. [63] inferred unlock patterns on smartphones via a notebook that is connected to the wireless hotspot provided by the smartphone. Li et al. [64] further improved these attacks by considering an attacker controlling only a WiFi access point. They infer the PIN input on smartphones and also analyze network packets to determine when the sensitive input starts.

4.3. Remote Side-Channel Attacks

The attacks presented in this section can be categorized as software-only attacks. In contrast to the local side-channel attacks as well as the vicinity side-channel attacks presented in the previous sections, these attacks neither require the attacker to be in the proximity nor in the vicinity of the targeted user. Hence, these attacks can be executed remotely and target a much larger scale.

Linux-inherited *procfs* Leaks. The Linux kernel releases "accounting" information that is considered as being harmless via the procfs. This includes, for example, the memory footprint (total virtual memory size and total physical memory size) of each application via `/proc/[pid]/statm`, CPU utilization times via `/proc/[pid]/stat`, number of context switches via `/proc/[pid]/status`, but also system-wide information like interrupt counters via `/proc/interrupts` and context switches via `/proc/stat`.

Attacks. Jana and Shmatikov [25] observed that the memory footprint of the browser correlates with the rendered website. Thus, by monitoring the memory footprint they can infer a user's browsing behavior (browser history), which represents sensitive information and is normally protected via the `READ_HISTORY_BOOKMARKS` permission. Later on, Chen et al. [65] exploited this information to detect *Activity* transitions within Android apps. They observed that the shared memory size increases by the size of the graphics buffer in both processes, *i.e.*, the app process and in the window compositor process (*SurfaceFlinger*). These increases occur due to the IPC communication between the app and the window manager. Besides, they also consider CPU utilization and network activity in order to infer the exact activity later on.

Similar to the memory footprint of applications, the procfs also provides system-wide information about the number of interrupts and context switches. Again, this information is considered as being innocuous and is, thus, published on purpose. Simon et al. [12] exploited this information to infer text entered via swipe input methods. Diao et al. [66] presented two attacks to infer unlock patterns and the app running in the foreground. The information leaks exploited were gathered from interrupt time series for the device's touchscreen controller. Besides, also the power consumption is released via the procfs. Yan et al. [32] have shown that the power consumption allows to infer the number of entered characters on the soft keyboard.

Data-Usage Statistics. Android keeps track of the amount of incoming and outgoing network traffic on a per-application basis. These statistics allow users to keep an eye on the data consumption of any app and can be accessed without any permission.

Attacks. Data-usage statistics are captured with a fine-grained granularity, *i.e.*, packet lengths of single TCP packets can be observed, and have already been successfully exploited. Zhou et al. [15] demonstrated that by monitoring the data-usage statistics an adversary can infer sensitive information of specific apps. They were able to infer disease conditions accessed via *WebMD*, the financial portfolio via *Yahoo! Finance*, and also a user's identity by observing the data-usage statistics of the *Twitter* app and exploiting the publicly available Twitter API. Later, it has been shown that the data-usage statistics can also be employed to fingerprint websites [26] even though the traffic is routed through the anonymity network Tor.

Page Deduplication. To reduce the overall memory footprint of a system, (some) operating systems[3] search for identical pages within the physical memory and merge them—even across different processes—which is called page deduplication. As soon as one process intends to write onto such a deduplicated page, a copy-on-write fault occurs and the process gets its own copy of this memory region again.

Attacks. Such copy-on-write faults have been exploited by Suzaki et al. [67] and recently Gruss et al. [68] demonstrated the possibility to measure the timing differences between normal write accesses and copy-on-write faults from within JavaScript code. Based on these precise timings they suggest to fingerprint visited websites by allocating memory that stores images found on popular websites. If the user browses the website with the corresponding image, then at some point the OS detects the identical content in the pages and deduplicates these pages. By continuously writing to the allocated memory, the attacker might observe a copy-on-write fault in which case the attacker knows that the user currently browses the corresponding website.

Cache Attacks. CPU caches represent an important component within the memory hierarchy of modern computer architectures. Multiple cache levels bridge the gap between the latency of main memory accesses and the fast CPU clock frequencies. However, by measuring execution times and memory accesses, an attacker can infer sensitive information from processes running in parallel on the same device [1].

Attacks. Cache-timing attacks against the AES have already been investigated on Android-based mobile devices. For instance, Bernstein's cache-timing attack [69] has been launched on development boards [70]–[72] and on Android smartphones [73], [74]. Besides, similar cache attacks have been launched on embedded devices [75] and more fine-grained attacks [5] against the AES have also been applied on smartphones [76]. These attacks relied on privileged access to precise timing measurements, but as stated by Oren et al. [77] cache attacks can also be exploited via JavaScript and, thus, do not require native code execution anymore. They even demonstrated the possibility to track user behavior including mouse movements as well as browsed websites via JavaScript-based cache attacks. A recent paper by Lipp et al. [78] even constitutes that all existing cache attacks, including the effective Flush+Reload attack [6], can be applied on modern Android-based smartphones without any privileges. While early attacks on smartphones exclusively targeted cryptographic implementations, their work also shows that user interactions (touch actions and swipe actions) can be inferred through this side channel. Similar investigations of Flush+Reload on ARM have also been conducted by Zhang et al. [79].

Sensor-based Keyloggers. Cai et al. [80] and Raij et al. [81] were one of the first to discuss possible privacy implications resulting from mobile devices equipped with cameras, microphones, GPS sensors, and motion sensors in general. Nevertheless, a category of attacks that received the most attention are sensor-based keyloggers. These attacks are based on two observations. First, smartphones are equipped with lots of sensors—both motion sensors as well as ambient sensors—that can be accessed without any permission, and second, these devices are operated with fingers while being held in the users' hands. Hence, the following attacks are all based on the observation that users tap/touch/swipe the touchscreen and that the device is slightly tilt and turned during the operation.

Attacks. In 2011, Cai and Chen [10] first investigated motion-based keylogging of single digits by exploiting the accelerometer sensor. Following this work, Owusu et al. [82] extended the attack to infer single characters and Aviv [11], [83] investigated the accelerometer to attack PIN and pattern inputs. Subsequent publications [84]–[86] also considered the combination of the accelerometer and the gyroscope in order to improve the performance as well as to infer even longer text inputs [87].

Since the W3C specifications allow access to the motion and orientation sensors from JavaScript, motion-based keylogging attacks have even been successfully demonstrated via websites [13], [88]. Even worse, some browsers continue to execute JavaScript code, although the user closed the browser or turned off the screen.

While the above summarized attacks exploit different types of motion sensors, e.g., accelerometer and gyroscope, keylogging attacks can also be employed by exploiting ambient sensors. Spreitzer [89] currently presented the only attack that exploits an ambient sensor, namely the ambient-light sensor, in order to infer a user's PIN input on touchscreens.

As demonstrated by Simon and Anderson [90], PIN inputs on smartphones can also be inferred by continuously taking pictures via the front camera. Afterwards, PIN digits can be inferred by image analysis and by investigating the relative changes of objects in subsequent pictures that correlate with the entered digits. Fiebig et al. [91] demon-

3. For instance, the Android-based CyanogenMod OS allows to enable page deduplication.

strated that the front camera can also be used to capture the screen reflections in the user's eyeballs, which also allows to infer user input. In a similar manner, Narain et al. [92] and Gupta et al. [93] showed that tap sounds (inaudible to the human ear) recorded via smartphone stereo-microphones can be used to infer typed text on the touchscreen. However, these attacks require the `CAMERA` and `RECORD_AUDIO` permission which might raise the user's suspicion during the installation. In contrast, the above presented motion and ambient sensors can be accessed without any permission.

For a more complete overview of sensor-based keylogging attacks we refer to the recently published survey papers by Hussain et al. [94] and Nahapetian [95]. Considering the significant number of papers that have been published in this context, user awareness about such attacks should be raised. Especially since a recent study by Mehrnezhad et al. [88] found that the perceived risk of motion sensors, and especially ambient sensors, among users is very low.

Fingerprinting Devices/Users. The identification of smartphones (and users) without a user's awareness is considered a privacy risk. While obvious identification mechanisms like device IDs and web cookies can be thwarted, hardware imperfections of hardware components, e.g., sensors, as well as specific software features can also be employed to stealthily fingerprint and identify devices and users, respectively.

Attacks. Bojinov et al. [96] and Dey et al. [97] observed that unique variations of sensor readings (e.g., accelerometer) can be used to fingerprint devices. These variations are a result of the manufacturing process and are persistent throughout the life of the sensor/device. As these sensors can also be accessed via JavaScript, it is possible to fingerprint devices via websites [98]. Similarly, such imperfections also affect the microphones and speakers [99], [100], which also allow to fingerprint devices. In addition, by combining multiple sensors even higher accuracies can be achieved [101].

Kurtz et al. [102] demonstrated that users can also be identified by fingerprinting mobile device configurations, e.g., device names, language settings, installed apps, etc. Hence, their fingerprinting approach exploits software properties (*i.e.*, configurations) only. Hupperich et al. [103] proposed to combine hardware as well as software features to fingerprint mobile devices.

Location Inference. As smartphones are always carried around, information about a phone's location inevitably reveals the user's location/position. Hence, resources that obviously can be used to determine a user's location, e.g., the GPS sensor, are considered as privacy relevant and, thus, require a dedicated permission. Yet, even without permissions, side-channel attacks can be used to infer precise location information.

Attacks. Han et al. [104], Nawaz et al. [105], and Narain et al. [106] demonstrated that the accelerometer and the gyroscope can be used to infer car driving routes. Similarly, Hemminki et al. [107] showed that the transportation mode, e.g., train, bus, metro, etc., can be inferred via the accelerometer readings of smartphones. Besides the accelerometer and the gyroscope also ambient sensors can be used to infer driving routes. Ho et al. [108] exploit the correlation between sensor readings of the barometer sensor and the geographic elevation to infer driving routes.

Even less obvious side-channels that allow to infer driving routes and locations are the speaker status information (e.g., speaker on/off) and the power consumption (available via the procfs). More specifically, Zhou et al. [15] observed that the Android API allows to query whether or not the speaker is currently active, *i.e.*, boolean information that indicates whether or not any app is playing sound on the speakers. They exploit this information to attack the turn-by-turn voice guidance of navigation systems. By continuously querying this API, they can determine how long the speaker is active. This information allows them to infer the speech length of voice direction elements, e.g., the length of "Turn right onto East Main Street". As driving routes consist of many such turn-by-turn voice guidances, they use this information to fingerprint driving routes. Michalevsky et al. [14] showed that the observed power consumption (via the procfs) is related to the strength of the cellular signal, which depends on the distance to the base station. Given this information they are able to infer a user's location.

Speech Recognition. Eavesdropping conversations represents a privacy threat. Thus, the `RECORD_AUDIO` permission protects access to the microphone. However, acoustic signals also influence the gyroscope measurements.

Attacks. Michalevsky et al. [109] exploited the gyroscope sensor to measure acoustic signals in the vicinity of the phone and to recover speech information. Although they only consider a small set of vocabulary, *i.e.*, digits only, their work demonstrates the immense power of gyroscope sensors in today's smartphones. By exploiting the gyroscope sensor they are able to bypass the `RECORD_AUDIO` permission.

Soundcomber. Interactive voice response systems supported by telephone services use dual-tone multi-frequency (DTMF) signaling to transmit entered numbers, *i.e.*, an audio signal is transmitted for each key.

Attacks. As DTMF tones are also played locally, Schlegel et al. [110] showed that by requesting the `RECORD_AUDIO` permission, these tones can be recorded and used to infer sensitive input like credit card numbers.

5. Active Attacks

Besides passively observing leaking information, an active attacker can also manipulate the target, its input, or its environment in order to subsequently observe leaking information via abnormal behavior of the target (cf. [4]).

5.1. Local Side-Channel Attacks

Most active attacks that require the attacker to be physically present with the attacked device have been investigated in the smartcard setting. Only few of these attacks are investigated on larger systems like smartphones.

Clock/Power Glitching. Variations of the clock signal, e.g., overclocking, have been shown to be an effective method for fault injection on embedded devices in the past. One prerequisite for this attack is an external clock source. Microcontrollers applied in smartphones typically have an internal clock generator making clock tampering impossible. Besides clock tampering, intended variations of the power supply represent an additional method for fault injection. With minor hardware modifications, power-supply tampering can be applied on most microcontroller platforms.

Attacks. In [111] it is shown how to disturb the program execution of an ARM CPU on a *Raspberry PI* by underpowering, *i.e.*, the supply voltage is set to GND for a short time. Tobich et al. [112] take advantage of the so-called *forward body bias injection* for inducing a fault during a RSA-CRT calculation. Due to the relatively easy application on modern microcontrollers, voltage-glitching attacks pose a serious threat for smartphones if attackers have physical access to the device. This has been demonstrated by O'Flynn [113] for an off-the-shelf Android smartphone.

Electromagnetic Fault Injection (EMFI). Transistors placed on microchips can be influenced by electromagnetic emanation. EMFI attacks take advantage of this fact. These attacks use short (in the range of nanoseconds), high-energy EM pulses to, e.g., change the state of memory cells resulting in erroneous calculations. In contrast to voltage glitching, where the injected fault is typically global, EMFI allows to target specific regions of a microchip by precisely placing the EM probe, e.g., on the instruction memory, the data memory, or CPU registers. Compared to optical fault injection, EMFI attacks do not necessarily require a decapsulation of the chip making them more practical.

Attacks. Ordas et al. [114] report successful EMFI attacks targeting the AES hardware module of a 32 bit ARM processor. Rivière et al. [115] use EMFI attacks to force instruction skips and instruction replacements on modern ARM microcontollers. Considering the fact that ARM processors are applied in modern smartphones, EMFI attacks represent a serious threat for such devices.

Laser/Optical Faults. Optical fault attacks using a laser beam are among the most-effective fault-injection techniques. These attacks take advantage of the fact that a focused laser beam can change the state of a transistor on a microcontroller resulting in, e.g., bit flips in memory cells. Compared to other fault-injection techniques (voltage glitching, EMFI), the effort for optical fault injection is high. (1) Decapsulation of the chip is a prerequisite in order to access the silicone with the laser beam. Besides, (2) finding the correct location for the laser beam to produce exploitable faults is also not a trivial task.

Attacks. First optical fault-injection attacks targeting an 8-bit microcontroller have been published by Skorobogatov and Anderson [116] in 2002. Inspired by their work, several optical fault-injection attacks have been published in the following years, most of them targeting smartcards or low-resource embedded devices (e.g. [117], [118]). The increasing number of metal layers on top of the silicone, decreasing feature size (small process technology), and the high decapsulation effort make optical fault injection difficult to apply on modern microprocessors used in smartphones.

NAND Mirroring. Data mirroring refers to the replication of data storage between different locations. Such techniques are used to recover critical data after disasters but also allow to restore a previous system state.

Attacks. The Apple iPhone protects a user's privacy by encrypting the data. Therefore, a passcode and a hardware-based key are used to derive various keys that can be used to protect the data on the device. As a dedicated hardware-based key is used to derive these keys, brute-force attempts must be done on the attacked device. Furthermore, brute-force attempts are discouraged by gradually increasing the waiting time between wrongly entered passcodes up to the point where the phone is wiped. In response to the Apple vs FBI case, Skorobogatov [119] demonstrated that NAND mirroring can be used to reset the phone state and, thus, can be used to brute-force the passcode. Clearly, this approach also represents an active attack as the attacker actively influences (resets) the state of the device.

Temperature Variation. Operating a device outside of its specified temperature range allows to cause faulty behavior. Heating up a device above the maximum specified temperature can cause faults in memory cells. Cooling down the device has an effect on the speed RAM content fades away after power off (*remanence effect* of RAM).

Attacks. Hutter and Schmidt [120] present heating fault attacks targeting an AVR microcontroller. They prove the practicability of this approach by successfully attacking an RSA implementation on named microcontroller. FROST [121], on the other hand, is a tool for recovering disc encryption keys from RAM on Android devices by means of cold-boot attacks. Here the authors take advantage of the increased time data in RAM remains valid after power off due to low temperature.

5.2. Vicinity Side-Channel Attacks

Besides passively observing leaking information, vicinity attacks can be improved by considering active attackers as demonstrated by the following example.

Network Traffic Analysis. Network traffic analysis has already been discussed in the context of passive side-channel attacks in Section 4. However, active attackers might learn additional information by actively influencing the transmitted packets, e.g., by delaying packets.

Attacks. He et al. [122] demonstrated that an active attacker, e.g., represented by ISPs, could delay HTTP requests from Tor users in order to increase the performance of website fingerprinting attacks.

5.3. Remote Side-Channel Attacks

An area of research that gains increasing attention among the scientific community are software-induced faults.

Rowhammer. The increasing density of memory cells requires the size of these cells to decrease, which in turn decreases the charging of single cells but also causes electromagnetic coupling effects between cells.

Attacks. Kim et al. [123] demonstrated that these observations can be used to induce hardware faults, *i.e.*, bit flips in neighboring cells, via frequent memory accesses to the main memory. Later, Seaborn and Dullien [124] demonstrated how to possibly exploit these bit flips from native code and Gruss et al. [125] showed that such bit flips can even be induced via JavaScript code. A recent paper [126] successfully demonstrates the exploitation of the Rowhammer bug to gain root privileges on Android smartphones by inducing bit flips from an unprivileged application.

6. Discussion of Countermeasures

In this section, we discuss existing countermeasures against the most prominent attacks. Overall we aim to shed light onto possible pitfalls of existing countermeasures and to stimulate future research to come up with more generic countermeasures against side-channel attacks.

6.1. Local Side-Channel Attacks

Protecting Cryptographic Implementations. Countermeasures from the smartcard world can be applied to protect cryptographic implementations on smartphones as well. Masking of sensitive values or execution randomization are countermeasures for hardening the implementation against passive attacks like power analysis or EM analysis [4]. Executing critical calculations twice allows to detect faults that are injected during an active side-channel attack [127].

Protecting User Input. Mitigation techniques to prevent attackers from inferring user input on touchscreens are not that thoroughly investigated yet. Nevertheless, proposed countermeasures include, for example, randomly starting the vibrator to prevent attacks that monitor the backside of the device [45], or to randomize the layout of the soft keyboard each time the user provides input to prevent smudge attacks [83] as well as attacks that monitor the hand movement [44]. Aviv [83] also proposes to align PIN digits in the middle of the screen and after each authentication the user needs to swipe down across all digits in order to hide smudges. Besides, Kwon and Na [128] introduce a new authentication mechanism denoted as *TinyLock* that should prevent smudge attacks against pattern unlock mechanisms. Krombholz et al. [129] proposed an authentication mechanism for devices with pressure-sensitive screens that should prevent smudge attacks and shoulder surfing attacks. Raguram et al. [41], [42] suggest to decrease the screen brightness, to disable visual feedback (e.g., pop-up characters) on soft keyboards, and to use anti-reflective coating in eyeglasses to prevent attackers from exploiting reflections.

6.2. Vicinity Side-Channel Attacks

Preventing Network Traffic Analysis. Countermeasures to prevent attackers from applying traffic analysis techniques on wiretapped network connections have been extensively considered in the context of website fingerprinting attacks. The main idea of these obfuscation techniques is to hide information that allows attackers to uniquely identify, e.g., visited websites. Proposed countermeasures [130]–[134], however, require the application as well as the remote server to cooperate. Furthermore, it has already been pointed out in [26] that these countermeasures add overhead in terms of bandwidth and data consumption which might not be acceptable in case of mobile devices with limited data plans.

6.3. Remote Side-Channel Attacks

Permissions. The most straight-forward approach always discussed as a viable means to prevent specific types of side-channel attacks is to protect the exploited information or resource by means of dedicated permissions. However, studies [135] have shown that permission-based approaches are not quite convincing. Some users do not understand the exact meaning of specific permissions, and others do not care about requested permissions. Acar et al. [16] even attest that the Android permission system "has failed in practice". Despite these problems it seems to be nearly impossible to add dedicated permissions for every exploited information.

Keyboard Layout Randomization. In order to prevent sensor-based keylogging attacks it has been suggested to randomize the keyboard layout of soft keyboards [82]. However, it remains an open question how this would affect usability and, intuitively, in case of randomized QWERTY keyboards it might make keyboard input nearly impossible.[4]

Limiting Access or Sampling Frequency. It has also been suggested to disable access to sensor readings during sensitive input or to reduce the sampling frequency of sensors. This, however, would prevent applications that heavily rely on sensor readings, e.g., pedometers.

Side-channel attacks like *Soundcomber* might be prevented by *AuDroid* [136], which is an extension to the SELinux reference monitor that has been integrated into Android to control access to system audio resources. As pointed out by the authors, there is no security mechanism in place for the host OS to control access to a mobile device's speakers, thus allowing untrusted apps to exploit this communication channel. *AuDroid* enforces security policies that prevent data in system apps and services from being leaked to (or used by) untrusted parties.

Noise Injection. Randomly starting the phone vibrator has been suggested [82] to prevent sensor-based keyloggers that exploit the accelerometer sensor. However, Shrestha et al. [137] showed that random vibrations do not provide protection. As an alternative, Shrestha et al. proposed a tool named *Slogger* that introduces noise into sensor readings

4. The Android-based CyanogenMod OS allows to enable such a feature for PIN inputs optionally.

as soon as the soft keyboard is running. In order to do so, Slogger relies on a tool that needs to be started via the ADB shell (in order to be executed with ADB capabilities). Slogger injects events into the files corresponding to the accelerometer and the gyroscope located in `/dev/input/`, which is why ADB privileges are required for this defense mechanism. The authors even evaluated the effectiveness of Slogger against two sensor-based keyloggers and found that the accuracy of sensor-based keyloggers can be reduced significantly. Das et al. [98] also suggest to add noise to sensor readings in order to prevent device fingerprinting via hardware imperfections of sensors. A more general approach that targets the injection of noise into the information provided via the procfs has been proposed by Xiao et al. [24].

Preventing Microarchitectural Attacks. The inherent nature of modern computer architectures allows for sophisticated attacks due to shared resources and especially due to dedicated performance optimization techniques. A famous and popular example is the memory hierarchy that introduces significant performance gains but also allows for microarchitectural attacks like cache attacks. Although specific cryptographic implementations can be protected against such attacks, e.g., bit-sliced implementations or dedicated hardware instructions can be used to protect AES implementations, generic countermeasures against cache attacks represent a non-trivial challenge. However, we consider it of utmost importance to spur further research in the context of countermeasures, especially since cache attacks do not only pose a risk for cryptographic algorithms, but also for other sensitive information like keystroke logging [7], [78].

App Guardian. Most of the above presented countermeasures aim to prevent very specific attacks only, but cannot be applied to prevent attacks within a specific category of our classification system, e.g., software-only attacks located in the upper right of our new classification system (cf. Figure 5). At least some of these attacks, however, have been addressed by App Guardian [17], which represents a more general approach to defend against software-only attacks. App Guardian is a third party application that runs in user mode and basically employs side-channel information to detect RIG attacks (including software-only side-channel attacks). The basic idea of App Guardian is to stop the malicious application while the principal (the app to be protected) is being executed and to resume the (potentially malicious) application later on. Although App Guardian still faces challenges it is a novel idea to cope with such side-channel attacks in general. More specifically, it tries to cope with all passive attacks that require the attacker to execute software on the targeted device (cf. Figure 5).

App Guardian seems to be a promising research project to cope with side-channel attacks on smartphones at a larger scale. However, an unsolved issue of App Guardian is the problem that it still struggles with the proper identification of applications to be protected. The effectiveness of App Guardian should be further evaluated against existing side-channel attacks and it might be interesting to extend it to cope with side-channel attacks conducted from within the browser, *i.e.*, to mitigate side-channel attacks via JavaScript.

7. Classification and Trend Analysis

In Figure 6 we classify the attacks surveyed in Section 4 and Section 5 according to our new classification system. Based on this classification system we observe specific trends in modern side-channel attacks that will be discussed within the following paragraphs. This trend analysis also includes pointers for possible research directions.

From Local to Remote Attacks. The first trend that can be observed is that, in contrast to the smartcard era, the smartphone era faces a shift towards remote side-channel attacks that focus on both hardware properties and software features. The shift from local attacks (during the smartcard era) towards remote attacks (on mobile devices) can be addressed to the fact that the attack scenario as well as the attacker have changed significantly. More specifically, side-channel attacks against smartcards have been conducted to reveal sensitive information that should be protected from being accessed by benign users. For example, in case of pay-TV cards the secret keys must be protected against benign users, *i.e.*, users who bought these pay-TV cards in the first place. The attacker in this case might be willing to invest in equipment in order to reveal the secret key as this key could be sold later on. In contrast, today's smartphones are used to store and process sensitive information and attackers interested in this information are usually not the users themselves but rather criminals, imposters, and other malicious entities that aim to steal this sensitive information from users. Especially the appification of the mobile ecosystem provides tremendous opportunities for attackers to exploit identified side-channel leaks via software-only attacks. Hence, this shift also significantly increases the scale at which attacks are conducted. While local attacks only target a few devices, remote attacks can be conducted on millions of devices at the same time by distributing software via available app markets.

From Active to Passive Attacks. The second trend that can be observed is that fault injection attacks have been quite popular on smartcards, whereas such (local) fault attacks are not that widely investigated on smartphones, at least at the moment. Consequently, we also observe that the variety of fault attacks conducted in the smartcard era has decreased significantly in the smartphone era, which can be addressed to the following observations. First, the targeted device itself, e.g., a smartphone, is far more expensive than a smartcard and, hence, fault attacks that potentially permanently break the device are only acceptable for very targeted attacks. Even in case of highly targeted attacks (cf. Apple vs FBI dispute) zero-day vulnerabilities might be chosen instead of local fault attacks.[5] Second, remote fault attacks seem to be harder to conduct as such faults are harder to induce via software execution. Currently, the only remote fault attack (also known as software-induced fault attack) is the Rowhammer attack, which however gets increasing attention among the scientific community and

5. However, in September 2016 Skorobogatov [119] demonstrated that NAND mirroring allows to bypass the PIN entry limit on the iPhone 5c.

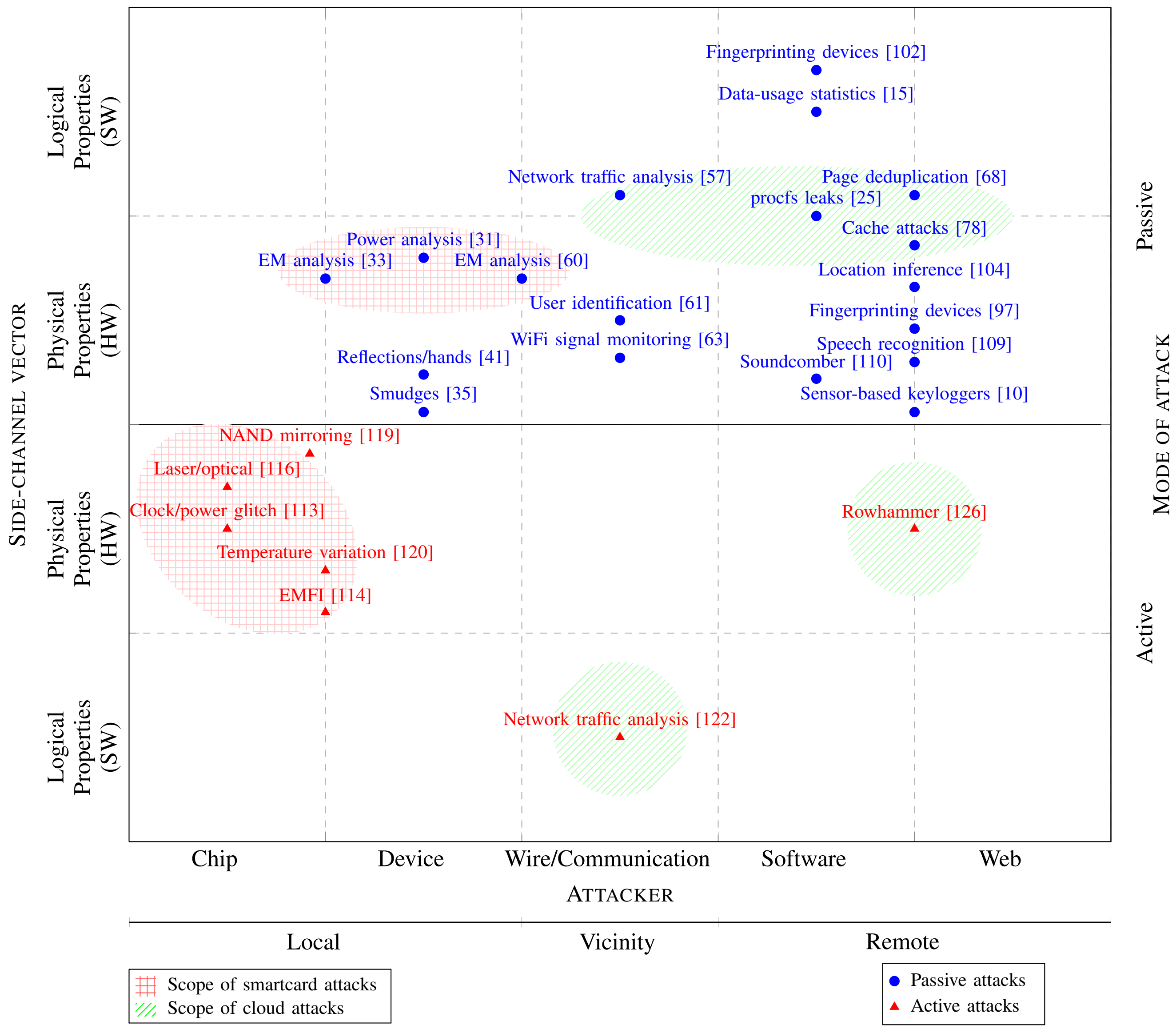

Figure 6. Classification of side-channel attacks: (1) active vs passive, (2) logical properties vs physical properties, (3) local vs vicinity vs remote.

has already been impressively exploited to gain root access on Android devices [126]. Although software-induced fault attacks have not been investigated extensively in the past, we expect further research to be conducted in this context.

Exploiting Physical and Logical Properties. In contrast to physical properties, logical properties (software features) do not result from any physical interaction with the device, but due to dedicated features provided via software. While traditional side-channel attacks mostly exploited physical properties and required dedicated equipment, more recent side-channel attacks exploit physical properties as well as logical properties. Interestingly, the immense number of sensors in smartphones also allows to exploit physical properties by means of software-only attacks, which was not possible on smartcards. Although the majority of attacks on mobile devices still exploits physical properties, the exploitation of logical properties also receives increasing attention. Especially the procfs seems to provide an almost inexhaustible source for possible information leaks. For example, the memory footprint released via the procfs has been used infer visited websites [25], or the number of context switches has been used to infer swipe input [12]. Besides, information that is available via official APIs is in some cases also available via the procfs like, e.g., the data-usage statistics that have been exploited to infer a user's identity [15] and to infer visited websites [26].

Empty Areas. As can be observed, a few areas in this categorization system are not (yet) covered. For instance, there is currently no active side-channel attack that exploits logical properties (software features) to induce faults. However, by considering existing passive attacks, one could come up with more advanced attacks by introducing an active attacker. Such an active attacker might, for example, block/influence a shared resource in order to cause malfunctioning of the target. For instance, considering the passive attack exploiting the speaker status (on/off) to infer a user's driving routes [15], one could easily induce faults by playing inaudible sounds in the right moment in order to prevent the turn-by-turn voice guidance from accessing the speaker. Thereby, the active attacker prevents the target (victim) from accessing the shared resource, *i.e.*, the speaker, and based on these induced "faults" an active attacker might gain an advantage compared to a passive attacker. We expect advances in this (yet) uncovered area of active side-channel attacks that target software features.

8. Issues, Challenges, and Future Research

In this section we discuss open issues and challenges that need to be addressed in future research. Hence, this section is not meant to provide solutions to existing problems. Instead, with the presented classification system for modern side-channel attacks we aim to shed light onto this vivid research area and, thereby, to point out high-level research directions. Overall, the ultimate goal is to spur further research in the context of side-channel attacks and countermeasures and, as a result, to pave the way for a more secure computing platform of smart and mobile devices.

Countermeasures. Side-channel attacks are published at an unprecedented pace and appropriate defense mechanisms are often either not (yet) available or cannot be deployed easily. Especially the five *key enablers* identified in this paper allow for devastating side-channel attacks that can be conducted remotely and, thus, target an immense number of devices and users at the same time. Although countermeasures are being researched, we observe a cat and mouse game between attackers and system engineers trying to make systems secure from a side-channel perspective. Besides, even if effective countermeasures were readily available, the mobile ecosystem of Android would impede a large-scale deployment of many of these defense mechanisms. The main problem is that even in case Google would apply defense mechanisms and patch these vulnerabilities, multiple device manufacturers as well as carriers also need to apply these patches to deploy countermeasures in practice. Hence, chances are that such countermeasures will never be deployed, especially not in case of outdated operating systems. We hope to stimulate research to come up with viable countermeasures in order to prevent such attacks at a larger scale, *i.e.*, by considering larger areas within the new categorization system, while also considering the challenges faced by the mobile ecosystem. For instance, App Guardian [17] follows the right direction by trying to cope with attacks at a larger scale, while at the same time it can be deployed as a third-party application.

Reproducibility and Responsible Disclosure. In order to foster research in the context of countermeasures, it would be helpful to publish the corresponding frameworks used to conduct side-channel attacks. While this might also address the long-standing problem of reproducibility of experiments in computer science in general, this would allow for a more efficient evaluation of developed countermeasures. At the same time, however, responsible disclosure must be upheld, which sometimes turns out to be a difficult balancing act. On the one hand, researchers want to publish their findings as soon as possible and on the other hand, putting countermeasures to practice might take some time.

Different Mobile Operating Systems and Cross-Platform Development. Research should not only focus on one particular OS exclusively, *i.e.*, especially Android seems to attract the most attention. Instead, the applicability of side-channel attacks should be investigated on multiple platforms as many (or most) of the existing attacks work on other platforms as well. This is due to the fact that different platforms and devices from different vendors aim to provide the same features like, for example, sensors and software interfaces, and rely on similar security concepts like permission systems and application sandboxing.

In addition, the increasing trend to develop applications for multiple platforms (cross-platform development) also increases the possibility to target multiple platforms at the same time. For example, the increasing popularity of HTML5 apps and the increasing availability of web APIs to access native resources from JavaScript significantly increases the scale of side-channel attacks as specific attacks possibly target multiple platforms at the same time.

Wearables. Although we put a strong focus on smartphones in this paper, we stress that wearables in general must be considered in future research. For example, smartwatches have already been employed to attack user input on POS terminals and hardware QWERTY keyboards [138]–[141]. Besides, it has also been demonstrated that smartwatches can be used to infer input on smartphones [142], [143] as well as text written on whiteboards [144]. With the ever increasing number of smart devices connected to our everyday life, the threat of side-channel attacks increases. We are likely to see higher accuracies when these attacks are performed across multiple devices, e.g., when combining data from smartwatches and smartphones. Furthermore, Farshteindiker et al. [145] also demonstrated how hardware implants (bugs)—possibly used by intelligence agencies—can be used to exfiltrate data by communicating with a smartphone. The communication channel is based on inaudible sounds emitted from the implant which are captured by the gyroscope of the smartphone. This interconnection clearly demonstrates the potential of attack vectors when multiple wearable devices are combined.

Internet of Things. Another area of research which is rapidly growing is the Internet of Things (IoT). As all devices in the IoT network are inter-connected and accessible via the Internet, we foresee that attackers will exploit side-channel leaks to target different kinds of IoT appliances. In fact such an attack has already been carried out by Zhang et al. [17]. They investigated an Android-based WiFi camera and observed that a particular side-channel leak on Android smartphones can be exploited to infer whether or not the user is at home. This example demonstrates that side-channel leaks do not only pose a threat to a user's privacy and security from a system security point of view, but also pose a threat to smart home appliances and security systems like, e.g., smart thermostats, cameras, and alarm systems. Although this sounds utopian at first, the above example clearly demonstrates that side-channel leaks (on smartphones) also pose a threat to these IoT appliances and puts even users' physical possessions at risk.

Combination of Multiple Information Leaks. In order to improve the accuracy of existing attacks or to come up with more sophisticated attack scenarios, multiple side-channel leaks can also be combined. For instance, the combination of cache attacks and sensor-based keyloggers as mentioned in [78] could be used to improve keylogging attacks. First, cache attacks can be used to determine the exact time when a key is entered and, second, sensor-based keyloggers can be used to infer the actual key. Furthermore, website fingerprinting attacks could be combined with sensor-based keyloggers as mentioned in [26], which would allow to steal login credentials for specific websites.

In addition, side-channel attacks can also be used to improve attacks that exploit software vulnerabilities. For example, although Screenmilker [27] does not represent a side-channel attack—because a software vulnerability is exploited—it relies on side-channel information in order to exploit this vulnerability in the right moment. They suggest to rely on CPU utilization, memory and network activities in order to determine whether the targeted app is executed and, thus, are able to take screenshots in the right moment.

Code Analysis Tools. The appification of mobile devices allows for easy download of apps from the app markets. However, these apps can be implemented by anyone who has a developer account and, thus, the code needs to be checked and verified appropriately, *i.e.*, for presence of malicious behavior and side-channel vectors. While the app vetting processes of app stores, e.g., Google Play, already check for presence of malware, dedicated technologies, such as static and dynamic code analysis, should also be employed in order to prevent apps prone to side-channel attacks and apps exploiting side-channel information leaks from being distributed via app markets. This, however, does not seem to be a trivial task since most side-channel attacks exploit information or resources that can be accessed without any specific privileges or permissions.

Besides, static and dynamic code analysis tools could help to fix delicate implementation flaws that lead to side-channel attacks. For instance, some implementation flaws exist for many years without being noticed as has been demonstrated in [146] for the OpenSSL implementation of the digital signature algorithm. Fostering the development and application of tools to find and detect such flaws during the software development process could help to prevent vulnerable code from being deployed.

9. Conclusion

Considering the immense threat arising from side-channel attacks on mobile devices, a thorough understanding of information leaks and possible exploitation techniques is necessary. Based on this open issue, we surveyed existing side-channel attacks and identified commonalities of these attacks in order to systematically categorize all existing attacks. With the presented classification system we aim to provide a thorough understanding of information leaks and hope to spur further research in the context of side-channel attacks as well as countermeasures and, thereby, to pave the way for secure computing platforms.

Acknowledgment

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644052 (HECTOR), and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402). Veelasha Moonsamy has been supported by the Technology Foundation STW (project 13499 - TYPHOON & ASPASIA) from the Dutch government. Further, we would like to thank Florian Mendel for helpful discussions about active side-channel attacks.

References

[1] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Advances in Cryptology – CRYPTO 1996*, ser. LNCS, vol. 1109. Springer, 1996, pp. 104–113.

[2] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology – CRYPTO 1999*, ser. LNCS, vol. 1666. Springer, 1999, pp. 388–397.

[3] J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards," in *Smart Card Programming and Security – E-smart 2001*, ser. LNCS, vol. 2140. Springer, 2001, pp. 200–210.

[4] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.

[5] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient Cache Attacks on AES, and Countermeasures," *J. Cryptology*, vol. 23, pp. 37–71, 2010.

[6] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *USENIX Security Symposium 2014*. USENIX Association, 2014, pp. 719–732.

[7] D. Gruss, R. Spreitzer, and S. Mangard, "Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches," in *USENIX Security Symposium 2015*. USENIX Association, 2015, pp. 897–912.

[8] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware," *IACR Cryptology ePrint Archive*, vol. 2016, p. 613, 2016.

[9] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in *USENIX Security Symposium 2016*. USENIX Association, 2016, pp. 565–581.

[10] L. Cai and H. Chen, "TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone Motion," in *USENIX Workshop on Hot Topics in Security – HotSec*. USENIX Association, 2011.

[11] A. J. Aviv, B. Sapp, M. Blaze, and J. M. Smith, "Practicality of Accelerometer Side Channels on Smartphones," in *Annual Computer Security Applications Conference – ACSAC 2012*. ACM, 2012, pp. 41–50.

[12] L. Simon, W. Xu, and R. Anderson, "Don't Interrupt Me While I Type: Inferring Text Entered Through Gesture Typing on Android Keyboards," *PoPETs*, vol. 2016, pp. 136–154, 2016.

[13] M. Mehrnezhad, E. Toreini, S. F. Shahandashti, and F. Hao, "TouchSignatures: Identification of User Touch Actions and PINs Based on Mobile Sensor Data via JavaScript," *J. Inf. Sec. Appl.*, vol. 26, pp. 23–38, 2016.

[14] Y. Michalevsky, A. Schulman, G. A. Veerapandian, D. Boneh, and G. Nakibly, "PowerSpy: Location Tracking Using Mobile Device Power Analysis," in *USENIX Security Symposium 2015*. USENIX Association, 2015, pp. 785–800.

[15] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt, "Identity, Location, Disease and More: Inferring Your Secrets From Android Public Resources," in *Conference on Computer and Communications Security – CCS 2013*. ACM, 2013, pp. 1017–1028.

[16] Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. D. McDaniel, and M. Smith, "SoK: Lessons Learned from Android Security Research for Appified Software Platforms," in *IEEE Symposium on Security and Privacy – S&P 2016*. IEEE, 2016, pp. 433–451.

[17] N. Zhang, K. Yuan, M. Naveed, X. Zhou, and X. Wang, "Leave Me Alone: App-Level Protection against Runtime Information Gathering on Android," in *IEEE Symposium on Security and Privacy – S&P 2015*. IEEE Computer Society, 2015, pp. 915–930.

[18] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp)iPhone: Decoding Vibrations From Nearby Keyboards Using Mobile Phone Accelerometers," in *Conference on Computer and Communications Security – CCS 2011*. ACM, 2011, pp. 551–562.

[19] T. Zhu, Q. Ma, S. Zhang, and Y. Liu, "Context-free Attacks Using Keyboard Acoustic Emanations," in *Conference on Computer and Communications Security – CCS 2014*. ACM, 2014, pp. 453–464.

[20] S. Biedermann, S. Katzenbeisser, and J. Szefer, "Hard Drive Side-Channel Attacks Using Smartphone Magnetic Field Sensors," in *Financial Cryptography – FC 2015*, ser. LNCS, vol. 8975. Springer, 2015, pp. 489–496.

[21] L. Schwittmann, V. Matkovic, M. Wander, and T. Weis, "Video Recognition Using Ambient Light Sensors," in *Pervasive Computing and Communication Workshops – PerCom 2016*. IEEE Computer Society, 2016, pp. 1–9.

[22] C. Song, F. Lin, Z. Ba, K. Ren, C. Zhou, and W. Xu, "My Smartphone Knows What You Print: Exploring Smartphone-based Side-channel Attacks Against 3D Printers," in *Conference on Computer and Communications Security – CCS 2016*. ACM, 2016, pp. 895–907.

[23] A. Hojjati, A. Adhikari, K. Struckmann, E. Chou, T. N. T. Nguyen, K. Madan, M. S. Winslett, C. A. Gunter, and W. P. King, "Leave Your Phone at the Door: Side Channels that Reveal Factory Floor Secrets," in *Conference on Computer and Communications Security – CCS 2016*. ACM, 2016, pp. 883–894.

[24] Q. Xiao, M. K. Reiter, and Y. Zhang, "Mitigating Storage Side Channels Using Statistical Privacy Mechanisms," in *Conference on Computer and Communications Security – CCS 2015*. ACM, 2015, pp. 1582–1594.

[25] S. Jana and V. Shmatikov, "Memento: Learning Secrets from Process Footprints," in *IEEE Symposium on Security and Privacy – S&P 2012*. IEEE Computer Society, 2012, pp. 143–157.

[26] R. Spreitzer, S. Griesmayr, T. Korak, and S. Mangard, "Exploiting Data-Usage Statistics for Website Fingerprinting Attacks on Android," in *Security and Privacy in Wireless and Mobile Networks – WISEC 2016*. ACM, 2016, pp. 49–60.

[27] C. Lin, H. Li, X. Zhou, and X. Wang, "Screenmilker: How to Milk Your Android Screen for Secrets," in *Network and Distributed System Security Symposium – NDSS 2014*. The Internet Society, 2014.

[28] J. Balasch, B. Gierlichs, O. Reparaz, and I. Verbauwhede, "DPA, Bitslicing and Masking at 1 GHz," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, ser. LNCS, vol. 9293. Springer, 2015, pp. 599–619.

[29] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom, "ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels," in *Conference on Computer and Communications Security – CCS 2016*. ACM, 2016, pp. 1626–1638.

[30] G. Goller and G. Sigl, "Side Channel Attacks on Smartphones and Embedded Devices Using Standard Radio Equipment," in *Constructive Side-Channel Analysis and Secure Design – COSADE 2015*, ser. LNCS, vol. 9064. Springer, 2015, pp. 255–270.

[31] P. Belgarric, P. Fouque, G. Macario-Rat, and M. Tibouchi, "Side-Channel Analysis of Weierstrass and Koblitz Curve ECDSA on Android Smartphones," in *Topics in Cryptology – CT-RSA 2016*, ser. LNCS, vol. 9610. Springer, 2016, pp. 236–252.

[32] L. Yan, Y. Guo, X. Chen, and H. Mei, "A Study on Power Side Channels on Mobile Devices," *CoRR*, vol. abs/1512.07972, 2015.

[33] C. H. Gebotys, S. Ho, and C. C. Tiu, "EM Analysis of Rijndael and ECC on a Wireless Java-Based PDA," in *Cryptographic Hardware and Embedded Systems – CHES 2005*, ser. LNCS, vol. 3659. Springer, 2005, pp. 250–264.

[34] Y. Nakano, Y. Souissi, R. Nguyen, L. Sauvage, J. Danger, S. Guilley, S. Kiyomoto, and Y. Miyake, "A Pre-processing Composition for Secret Key Recovery on Android Smartphone," in *Information Security Theory and Practice – WISTP 2014*, ser. LNCS, vol. 8501. Springer, 2014, pp. 76–91.

[35] A. J. Aviv, K. L. Gibson, E. Mossop, M. Blaze, and J. M. Smith, "Smudge Attacks on Smartphone Touch Screens," in *Workshop on Offensive Technologies – WOOT 2010*. USENIX Association, 2010.

[36] Y. Zhang, P. Xia, J. Luo, Z. Ling, B. Liu, and X. Fu, "Fingerprint Attack Against Touch-Enabled Devices," in *Security and Privacy in Smartphones & Mobile Devices – SPSM@CCS*. ACM, 2012, pp. 57–68.

[37] P. Andriotis, T. Tryfonas, G. C. Oikonomou, and C. Yildiz, "A Pilot Study on the Security of Pattern Screen-Lock Methods and Soft Side Channel Attacks," in *Security and Privacy in Wireless and Mobile Networks – WISEC 2013*. ACM, 2013, pp. 1–6.

[38] M. Backes, M. Dürmuth, and D. Unruh, "Compromising Reflections-or-How to Read LCD Monitors around the Corner," in *IEEE Symposium on Security and Privacy – S&P 2008*. IEEE Computer Society, 2008, pp. 158–169.

[39] M. Backes, T. Chen, M. Dürmuth, H. P. A. Lensch, and M. Welk, "Tempest in a Teapot: Compromising Reflections Revisited," in *IEEE Symposium on Security and Privacy – S&P 2009*. IEEE Computer Society, 2009, pp. 315–327.

[40] F. Maggi, S. Gasparini, and G. Boracchi, "A Fast Eavesdropping Attack Against Touchscreens," in *Information Assurance and Security – IAS 2011*. IEEE, 2011, pp. 320–325.

[41] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J. Frahm, "iSpy: Automatic Reconstruction of Typed Input from Compromising Reflections," in *Conference on Computer and Communications Security – CCS 2011*. ACM, 2011, pp. 527–536.

[42] R. Raguram, A. M. White, Y. Xu, J. Frahm, P. Georgel, and F. Monrose, "On the Privacy Risks of Virtual Keyboards: Automatic Reconstruction of Typed Input from Compromising Reflections," *IEEE Trans. Dependable Sec. Comput.*, vol. 10, pp. 154–167, 2013.

[43] Y. Xu, J. Heinly, A. M. White, F. Monrose, and J. Frahm, "Seeing Double: Reconstructing Obscured Typed Input from Repeated Compromising Reflections," in *Conference on Computer and Communications Security – CCS 2013*. ACM, 2013, pp. 1063–1074.

[44] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha, "Beware, Your Hands Reveal Your Secrets!" in *Conference on Computer and Communications Security – CCS 2014*. ACM, 2014, pp. 904–917.

[45] J. Sun, X. Jin, Y. Chen, J. Zhang, Y. Zhang, and R. Zhang, "VISIBLE: Video-Assisted Keystroke Inference from Tablet Backside Motion," in *Network and Distributed System Security Symposium – NDSS 2016*. The Internet Society, 2016.

[46] Q. Yue, Z. Ling, X. Fu, B. Liu, K. Ren, and W. Zhao, "Blind Recognition of Touched Keys on Mobile Devices," in *Conference on Computer and Communications Security – CCS 2014*. ACM, 2014, pp. 1403–1414.

[47] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching From a Distance: Website Fingerprinting Attacks and Defenses," in *Conference on Computer and Communications Security – CCS 2012*. ACM, 2012, pp. 605–616.

[48] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website Fingerprinting in Onion Routing Based Anonymization Networks," in *Workshop on Privacy in the Electronic Society – WPES 2011*. ACM, 2011, pp. 103–114.

[49] T. Wang and I. Goldberg, "Improved Website Fingerprinting on Tor," in *Workshop on Privacy in the Electronic Society – WPES 2013*. ACM, 2013, pp. 201–212.

[50] M. Juárez, S. Afroz, G. Acar, C. Díaz, and R. Greenstadt, "A Critical Evaluation of Website Fingerprinting Attacks," in *Conference on Computer and Communications Security – CCS 2014*. ACM, 2014, pp. 263–274.

[51] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website Fingerprinting at Internet Scale," in *Network and Distributed System Security Symposium – NDSS 2016*. The Internet Society, 2016.

[52] T. Stöber, M. Frank, J. B. Schmitt, and I. Martinovic, "Who Do You Sync You Are?: Smartphone Fingerprinting via Application Behaviour," in *Security and Privacy in Wireless and Mobile Networks – WISEC 2013*. ACM, 2013, pp. 7–12.

[53] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing Android Encrypted Network Traffic to Identify User Actions," *IEEE Trans. Information Forensics and Security*, vol. 11, pp. 114–125, 2016.

[54] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "NetworkProfiler: Towards Automatic Fingerprinting of Android Apps," in *IEEE INFOCOM 2013*. IEEE, 2013, pp. 809–817.

[55] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I Know What You Did on Your Smartphone: Inferring App Usage Over Encrypted Data Traffic," in *Communications and Network Security – CNS 2015*. IEEE, 2015, pp. 433–441.

[56] S. Miskovic, G. M. Lee, Y. Liao, and M. Baldi, "AppPrint: Automatic Fingerprinting of Mobile Applications in Network Traffic," in *Passive and Active Measurement – PAM 2015*, ser. LNCS, vol. 8995. Springer, 2015, pp. 57–69.

[57] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "AppScanner: Automatic Fingerprinting of Smartphone Apps from Encrypted Network Traffic," in *IEEE European Symposium on Security and Privacy – EURO S&P 2016*. IEEE, 2016, pp. 439–454.

[58] H. F. Alan and J. Kaur, "Can Android Applications Be Identified Using Only TCP/IP Headers of Their Launch Time Traffic?" in *Security and Privacy in Wireless and Mobile Networks – WISEC 2016*. ACM, 2016, pp. 61–66.

[59] B. Saltaformaggio, H. Choi, K. Johnson, Y. Kwon, Q. Zhang, X. Zhang, D. Xu, and J. Qian, "Eavesdropping on Fine-Grained User Activities Within Smartphone Apps Over Encrypted Network Traffic," in *Workshop on Offensive Technologies – WOOT 2016*. USENIX Association, 2016.

[60] M. Schulz, P. Klapper, M. Hollick, E. Tews, and S. Katzenbeisser, "Trust The Wire, They Always Told Me!: On Practical Non-Destructive Wire-Tap Attacks Against Ethernet," in *Security and Privacy in Wireless and Mobile Networks – WISEC 2016*. ACM, 2016, pp. 43–48.

[61] M. Conti, M. Nati, E. Rotundo, and R. Spolaor, "Mind The Plug! Laptop-User Recognition Through Power Consumption," in *Workshop on IoT Privacy, Trust, and Security – IoTPTS@AsiaCCS*. ACM, 2016, pp. 37–44.

[62] K. Ali, A. X. Liu, W. Wang, and M. Shahzad, "Keystroke Recognition Using WiFi Signals," in *Mobile Computing and Networking – MOBICOM 2015*. ACM, 2015, pp. 90–102.

[63] J. Zhang, X. Zheng, Z. Tang, T. Xing, X. Chen, D. Fang, R. Li, X. Gong, and F. Chen, "Privacy Leakage in Mobile Sensing: Your Unlock Passwords Can Be Leaked through Wireless Hotspot Functionality," *Mobile Information Systems*, vol. 2016, pp. 8 793 025:1–8 793 025:14, 2016.

[64] M. Li, Y. Meng, J. Liu, H. Zhu, X. Liang, Y. Liu, and N. Ruan, "When CSI Meets Public WiFi: Inferring Your Mobile Phone Password via WiFi Signals," in *Conference on Computer and Communications Security – CCS 2016*. ACM, 2016, pp. 1068–1079.

[65] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into Your App without Actually Seeing It: UI State Inference and Novel Android Attacks," in *USENIX Security Symposium 2014*. USENIX Association, 2014, pp. 1037–1052.

[66] W. Diao, X. Liu, Z. Li, and K. 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*. IEEE, 2016, pp. 414–432.

[67] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, "Memory Deduplication as a Threat to the Guest OS," in *European Workshop on System Security – EUROSEC 2011*. ACM, 2011, p. 1.

[68] D. Gruss, D. Bidner, and S. Mangard, "Practical Memory Deduplication Attacks in Sandboxed Javascript," in *European Symposium on Research in Computer Security – ESORICS 2015*, ser. LNCS, vol. 9327. Springer, 2015, pp. 108–122.

[69] D. J. Bernstein, "Cache-Timing Attacks on AES," 2004, URL: `http://cr.yp.to/papers.html#cachetiming`.

[70] M. Weiß, B. Heinz, and F. Stumpf, "A Cache Timing Attack on AES in Virtualization Environments," in *Financial Cryptography – FC 2012*, ser. LNCS, vol. 7397. Springer, 2012, pp. 314–328.

[71] M. Weiß, B. Weggenmann, M. August, and G. Sigl, "On Cache Timing Attacks Considering Multi-core Aspects in Virtualized Embedded Systems," in *Conference on Trusted Systems – INTRUST 2014*, ser. LNCS, vol. 9473. Springer, 2014, pp. 151–167.

[72] A. Zankl, K. Miller, J. Heyszl, and G. Sigl, "Towards Efficient Evaluation of a Time-Driven Cache Attack on Modern Processors," in *European Symposium on Research in Computer Security – ESORICS 2016*, ser. LNCS, vol. 9879. Springer, 2016, pp. 3–19.

[73] R. Spreitzer and T. Plos, "On the Applicability of Time-Driven Cache Attacks on Mobile Devices," in *Network and System Security – NSS 2013*, ser. LNCS, vol. 7873. Springer, 2013, pp. 656–662.

[74] R. Spreitzer and B. Gérard, "Towards More Practical Time-Driven Cache Attacks," in *Information Security Theory and Practice – WISTP 2014*, ser. LNCS, vol. 8501. Springer, 2014, pp. 24–39.

[75] A. Bogdanov, T. Eisenbarth, C. Paar, and M. Wienecke, "Differential Cache-Collision Timing Attacks on AES with Applications to Embedded CPUs," in *Topics in Cryptology – CT-RSA 2010*, ser. LNCS, vol. 5985. Springer, 2010, pp. 235–251.

[76] R. Spreitzer and T. Plos, "Cache-Access Pattern Attack on Disaligned AES T-Tables," in *Constructive Side-Channel Analysis and Secure Design – COSADE 2013*, ser. LNCS, vol. 7864. Springer, 2013, pp. 200–214.

[77] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications," in *Conference on Computer and Communications Security – CCS 2015*. ACM, 2015, pp. 1406–1418.

[78] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache Attacks on Mobile Devices," in *USENIX Security Symposium 2016*. USENIX Association, 2016, pp. 549–564.

[79] X. Zhang, Y. Xiao, and Y. Zhang, "Return-Oriented Flush-Reload Side Channels on ARM and Their Implications for Android Devices," in *Conference on Computer and Communications Security – CCS 2016*. ACM, 2016, pp. 858–870.

[80] L. Cai, S. Machiraju, and H. Chen, "Defending Against Sensor-Sniffing Attacks on Mobile Phones," in *Workshop on Networking, Systems, and Applications for Mobile Handhelds – MobiHeld*. ACM, 2009, pp. 31–36.

[81] A. Raij, A. Ghosh, S. Kumar, and M. B. Srivastava, "Privacy Risks Emerging from the Adoption of Innocuous Wearable Sensors in the Mobile Environment," in *Conference on Human Factors in Computing Systems – CHI 2011*. ACM, 2011, pp. 11–20.

[82] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "ACCessory: Password Inference Using Accelerometers on Smartphones," in *Mobile Computing Systems and Applications – HotMobile 2012*. ACM, 2012, p. 9.

[83] A. J. Aviv, "Side Channels Enable By Smartphone Interaction," Ph.D. dissertation, University of Pennsylvania, 2012.

[84] Z. Xu, K. Bai, and S. Zhu, "TapLogger: Inferring User Inputs on Smartphone Touchscreens Using On-Board Motion Sensors," in *Security and Privacy in Wireless and Mobile Networks – WISEC 2012*. ACM, 2012, pp. 113–124.

[85] L. Cai and H. Chen, "On the Practicality of Motion Based Keystroke Inference Attack," in *Trust and Trustworthy Computing – TRUST 2012*, ser. LNCS, vol. 7344. Springer, 2012, pp. 273–290.

[86] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: Your Finger Taps Have Fingerprints," in *Mobile Systems – MobiSys 2012*. ACM, 2012, pp. 323–336.

[87] D. Ping, X. Sun, and B. Mao, "TextLogger: Inferring Longer Inputs on Touch Screen Using Motion Sensors," in *Security and Privacy in Wireless and Mobile Networks – WISEC 2015*. ACM, 2015, pp. 24:1–24:12.

[88] M. Mehrnezhad, E. Toreini, S. F. Shahandashti, and F. Hao, "Stealing PINs via Mobile Sensors: Actual Risk versus User Perception," *CoRR*, vol. abs/1605.05549, 2016.

[89] R. Spreitzer, "PIN Skimming: Exploiting the Ambient-Light Sensor in Mobile Devices," in *Security and Privacy in Smartphones & Mobile Devices – SPSM@CCS*. ACM, 2014, pp. 51–62.

[90] L. Simon and R. Anderson, "PIN Skimmer: Inferring PINs Through the Camera and Microphone," in *Security and Privacy in Smartphones & Mobile Devices – SPSM@CCS*. ACM, 2013, pp. 67–78.

[91] T. Fiebig, J. Krissler, and R. Hänsch, "Security Impact of High Resolution Smartphone Cameras," in *Workshop on Offensive Technologies – WOOT 2014*. USENIX Association, 2014.

[92] S. Narain, A. Sanatinia, and G. Noubir, "Single-Stroke Language-Agnostic Keylogging Using Stereo-Microphones and Domain Specific Machine Learning," in *Security and Privacy in Wireless and Mobile Networks – WISEC 2014*. ACM, 2014, pp. 201–212.

[93] H. Gupta, S. Sural, V. Atluri, and J. Vaidya, "Deciphering Text from Touchscreen Key Taps," in *Data and Applications Security and Privacy – DBSec 2016*, ser. LNCS, vol. 9766. Springer, 2016, pp. 3–18.

[94] M. Hussain, A. Al-Haiqi, A. A. Zaidan, B. B. Zaidan, M. L. M. Kiah, N. B. Anuar, and M. Abdulnabi, "The Rise of Keyloggers on Smartphones: A Survey and Insight Into Motion-Based Tap Inference Attacks," *Pervasive and Mobile Computing*, vol. 25, pp. 1–25, 2016.

[95] A. Nahapetian, "Side-Channel Attacks on Mobile and Wearable Systems," in *Consumer Communications & Networking Conference – CCNC 2016*. IEEE, 2016, pp. 243–247.

[96] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh, "Mobile Device Identification via Sensor Fingerprinting," *CoRR*, vol. abs/1408.1416, 2014.

[97] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, "AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable," in *Network and Distributed System Security Symposium – NDSS 2014*. The Internet Society, 2014.

[98] A. Das, N. Borisov, and M. Caesar, "Tracking Mobile Web Users Through Motion Sensors: Attacks and Defenses," in *Network and Distributed System Security Symposium – NDSS 2016*. The Internet Society, 2016.

[99] ——, "Do You Hear What I Hear?: Fingerprinting Smart Devices Through Embedded Acoustic Components," in *Conference on Computer and Communications Security – CCS 2014*. ACM, 2014, pp. 441–452.

[100] Z. Zhou, W. Diao, X. Liu, and K. Zhang, "Acoustic Fingerprinting Revisited: Generate Stable Device ID Stealthily with Inaudible Sound," in *Conference on Computer and Communications Security – CCS 2014*. ACM, 2014, pp. 429–440.

[101] T. Hupperich, H. Hosseini, and T. Holz, "Leveraging Sensor Fingerprinting for Mobile Device Authentication," in *Detection of Intrusions and Malware & Vulnerability Assessment – DIMVA 2016*, ser. LNCS, vol. 9721. Springer, 2016, pp. 377–396.

[102] A. Kurtz, H. Gascon, T. Becker, K. Rieck, and F. C. Freiling, "Fingerprinting Mobile Devices Using Personalized Configurations," *PoPETs*, vol. 2016, pp. 4–19, 2016.

[103] T. Hupperich, D. Maiorca, M. Kührer, T. Holz, and G. Giacinto, "On the Robustness of Mobile Device Fingerprinting: Can Mobile Users Escape Modern Web-Tracking Mechanisms?" in *Annual Computer Security Applications Conference – ACSAC 2015*. ACM, 2015, pp. 191–200.

[104] J. Han, E. Owusu, L. T. Nguyen, A. Perrig, and J. Zhang, "ACComplice: Location Inference Using Accelerometers on Smartphones," in *International Conference on Communication Systems and Networks – COMSNETS 2012*. IEEE, 2012, pp. 1–9.

[105] S. Nawaz and C. Mascolo, "Mining Users' Significant Driving Routes with Low-Power Sensors," in *Conference on Embedded Network Sensor Systems – SenSys 2014*. ACM, 2014, pp. 236–250.

[106] S. Narain, T. D. Vo-Huu, K. Block, and G. Noubir, "Inferring User Routes and Locations Using Zero-Permission Mobile Sensors," in *IEEE Symposium on Security and Privacy – S&P 2016*. IEEE, 2016, pp. 397–413.

[107] S. Hemminki, P. Nurmi, and S. Tarkoma, "Accelerometer-Based Transportation Mode Detection on Smartphones," in *Conference on Embedded Network Sensor Systems – SenSys 2013*. ACM, 2013, pp. 13:1–13:14.

[108] B. Ho, P. D. Martin, P. Swaminathan, and M. B. Srivastava, "From Pressure to Path: Barometer-based Vehicle Tracking," in *Embedded Systems for Energy-Efficient Built Environments – BuildSys*. ACM, 2015, pp. 65–74.

[109] Y. Michalevsky, D. Boneh, and G. Nakibly, "Gyrophone: Recognizing Speech from Gyroscope Signals," in *USENIX Security Symposium 2014*. USENIX Association, 2014, pp. 1053–1067.

[110] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones," in *Network and Distributed System Security Symposium – NDSS 2011*. The Internet Society, 2011.

[111] NewAE Technology Inc., "Fault Injection Raspberry PI," https://wiki.newae.com, accessed: 2016-08-03.

[112] K. Tobich, P. Maurine, P. Liardet, M. Lisart, and T. Ordas, "Voltage Spikes on the Substrate to Obtain Timing Faults," in *Digital System Design – DSD 2013*. IEEE Computer Society, 2013, pp. 483–486.

[113] C. O'Flynn, "Fault Injection using Crowbars on Embedded Systems," *IACR Cryptology ePrint Archive*, vol. 2016, p. 810, 2016.

[114] S. Ordas, L. Guillaume-Sage, and P. Maurine, "Electromagnetic Fault Injection: The Curse of Flip-Flops," *Journal of Cryptographic Engineering*, pp. 1–15, 2016.

[115] L. Rivière, Z. Najm, P. Rauzy, J. Danger, J. Bringer, and L. Sauvage, "High Precision Fault Injections on the Instruction Cache of ARMv7-M Architectures," in *Hardware Oriented Security and Trust – HOST 2015*. IEEE Computer Society, 2015, pp. 62–67.

[116] S. P. Skorobogatov and R. J. Anderson, "Optical Fault Induction Attacks," in *Cryptographic Hardware and Embedded Systems – CHES 2002*, ser. LNCS, vol. 2523. Springer, 2002, pp. 2–12.

[117] J. G. J. van Woudenberg, M. F. Witteman, and F. Menarini, "Practical Optical Fault Injection on Secure Microcontrollers," in *Fault Diagnosis and Tolerance in Cryptography – FDTC 2011*. IEEE Computer Society, 2011, pp. 91–99.

[118] C. Roscian, A. Sarafianos, J. Dutertre, and A. Tria, "Fault Model Analysis of Laser-Induced Faults in SRAM Memory Cells," in *Fault Diagnosis and Tolerance in Cryptography – FDTC 2013*. IEEE Computer Society, 2013, pp. 89–98.

[119] S. Skorobogatov, "The Bumpy Road Towards iPhone 5c NAND Mirroring," *CoRR*, vol. abs/1609.04327, 2016.

[120] M. Hutter and J. Schmidt, "The Temperature Side Channel and Heating Fault Attacks," in *Smart Card Research and Advanced Applications – CARDIS 2013*, ser. LNCS, vol. 8419. Springer, 2013, pp. 219–235.

[121] T. Müller and M. Spreitzenbarth, "FROST - Forensic Recovery of Scrambled Telephones," in *Applied Cryptography and Network Security – ACNS 2013*, ser. LNCS, vol. 7954. Springer, 2013, pp. 373–388.

[122] G. He, M. Yang, X. Gu, J. Luo, and Y. Ma, "A Novel Active Website Fingerprinting Attack Against Tor Anonymous System," in *Computer Supported Cooperative Work in Design – CSCWD 2014*. IEEE, 2014, pp. 112–117.

[123] Y. Kim, R. Daly, J. Kim, C. Fallin, J. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *International Symposium on Computer Architecture – ISCA 2014*. IEEE Computer Society, 2014, pp. 361–372.

[124] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," Blackhat 2015.

[125] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript," in *Detection of Intrusions and Malware & Vulnerability Assessment – DIMVA 2016*, ser. LNCS, vol. 9721. Springer, 2016, pp. 300–321.

[126] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *Conference on Computer and Communications Security – CCS 2016*. ACM, 2016, pp. 1675–1689.

[127] V. Lomné, T. Roche, and A. Thillard, "On the Need of Randomness in Fault Attack Countermeasures - Application to AES," in *Fault Diagnosis and Tolerance in Cryptography – FDTC 2012*. IEEE Computer Society, 2012, pp. 85–94.

[128] T. Kwon and S. Na, "TinyLock: Affordable Defense Against Smudge Attacks on Smartphone Pattern Lock Systems," *Computers & Security*, vol. 42, pp. 137–150, 2014.

[129] K. Krombholz, T. Hupperich, and T. Holz, "Use the Force: Evaluating Force-Sensitive Authentication for Mobile Devices," in *Symposium On Usable Privacy and Security – SOUPS 2016*. USENIX Association, 2016, pp. 207–219.

[130] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis," in *Network and Distributed System Security Symposium – NDSS 2009*. The Internet Society, 2009.

[131] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci, "HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows," in *Network and Distributed System Security Symposium – NDSS 2011*. The Internet Society, 2011.

[132] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail," in *IEEE Symposium on Security and Privacy – S&P 2012*. IEEE Computer Society, 2012, pp. 332–346.

[133] X. Cai, R. Nithyanand, and R. Johnson, "CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense," in *Workshop on Privacy in the Electronic Society – WPES 2014*. ACM, 2014, pp. 121–130.

[134] R. Nithyanand, X. Cai, and R. Johnson, "Glove: A Bespoke Website Fingerprinting Defense," in *Workshop on Privacy in the Electronic Society – WPES 2014*. ACM, 2014, pp. 131–134.

[135] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android Permissions: User Attention, Comprehension, and Behavior," in *Symposium On Usable Privacy and Security – SOUPS 2012*. ACM, 2012, p. 3.

[136] G. Petracca, Y. Sun, T. Jaeger, and A. Atamli, "AuDroid: Preventing Attacks on Audio Channels in Mobile Devices," in *Annual Computer Security Applications Conference – ACSAC 2015*. ACM, 2015, pp. 181–190.

[137] P. Shrestha, M. Mohamed, and N. Saxena, "Slogger: Smashing Motion-based Touchstroke Logging with Transparent System Noise," in *Security and Privacy in Wireless and Mobile Networks – WISEC 2016*. ACM, 2016, pp. 67–77.

[138] H. Wang, T. T. Lai, and R. R. Choudhury, "MoLe: Motion Leaks through Smartwatch Sensors," in *Mobile Computing and Networking – MOBICOM 2015*. ACM, 2015, pp. 155–166.

[139] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When Good Becomes Evil: Keystroke Inference with Smartwatch," in *Conference on Computer and Communications Security – CCS 2015*. ACM, 2015, pp. 1273–1285.

[140] A. Maiti, O. Armbruster, M. Jadliwala, and J. He, "Smartwatch-Based Keystroke Inference Attacks and Context-Aware Protection Mechanisms," in *Asia Conference on Computer and Communications Security – AsiaCCS*. ACM, 2016, pp. 795–806.

[141] C. Wang, X. Guo, Y. Wang, Y. Chen, and B. Liu, "Friend or Foe?: Your Wearable Devices Reveal Your Personal PIN," in *Asia Conference on Computer and Communications Security – AsiaCCS*. ACM, 2016, pp. 189–200.

[142] A. Sarkisyan, R. Debbiny, and A. Nahapetian, "WristSnoop: Smartphone PINs Prediction Using Smartwatch Motion Sensors," in *Workshop on Information Forensics and Security – WIFS 2015*. IEEE, 2015, pp. 1–6.

[143] A. Maiti, M. Jadliwala, J. He, and I. Bilogrevic, "(Smart)Watch Your Taps: Side-Channel Keystroke Inference Attacks using Smartwatches," in *International Symposium on Wearable Computers – ISWC 2015*. ACM, 2015, pp. 27–30.

[144] L. Arduser, P. Bissig, P. Brandes, and R. Wattenhofer, "Recognizing Text Using Motion Data From a Smartwatch," in *Pervasive Computing and Communication Workshops – PerCom 2016*. IEEE Computer Society, 2016, pp. 1–6.

[145] B. Farshteindiker, N. Hasidim, A. Grosz, and Y. Oren, "How to Phone Home with Someone Else's Phone: Information Exfiltration Using Intentional Sound Noise on Gyroscopic Sensors," in *Workshop on Offensive Technologies – WOOT 2016*. USENIX Association, 2016.

[146] C. P. García, B. B. Brumley, and Y. Yarom, ""Make Sure DSA Signing Exponentiations Really are Constant-Time"," in *Conference on Computer and Communications Security – CCS 2016*. ACM, 2016, pp. 1639–1650.