CovertPower

A Covert Channel on Android Devices Through USB Power Line

Spolaor, Riccardo; Xu, Yi; Moonsamy, Veelasha; Conti, Mauro; Cheng, Xiuzhen

DOI
10.1109/TDSC.2025.3556468

Publication date
2025

Document Version
Final published version

Published in
IEEE Transactions on Dependable and Secure Computing

Citation (APA)
Spolaor, R., Xu, Y., Moonsamy, V., Conti, M., & Cheng, X. (2025). CovertPower: A Covert Channel on Android Devices Through USB Power Line. *IEEE Transactions on Dependable and Secure Computing*, *22*(5), 4911-4926. https://doi.org/10.1109/TDSC.2025.3556468

CovertPower: A Covert Channel on Android Devices Through USB Power Line

Riccardo Spolaor, *Member, IEEE*, Yi Xu, Veelasha Moonsamy, Mauro Conti, *Fellow, IEEE*, and Xiuzhen Cheng, *Fellow, IEEE*

***Abstract*—Android operating system restricts access to data by enabling data control flow and permission systems to reduce the risk of information theft. Therefore, attackers are constantly looking for alternative and stealthy approaches to exfiltrate private data from a targeted device. This paper presents CovertPower, a covert channel attack that exfiltrates user data by actively inducing power consumption on Android devices. At the transmitting end, our CovertPower app modulates binary data into a timed resource workload (e.g., processor, write-on-memory), producing power consumption bursts. On the receiving end, we acquire power consumption traces via a low-cost hardware tool that can be easily concealed in USB wall-socket adapters or powerbanks. Therefore, a signal processing-based decoder analyzes such traces and retrieves the exfiltrated information. We demonstrate the feasibility of our attack with a thorough experimental evaluation on 14 mobile devices and various real-world settings such as display state, ongoing activities, and charging technologies. Our attack achieves a transfer speed of up to 10 bps with a high bit sequence similarity on most devices and settings considered.**

***Index Terms*—Covert channel, power consumption, data exfiltration, mobile devices, Android, USB power line.**

I. Introduction

SECURITY experts have developed measures that enforce information flow policies and network restrictions to prevent the theft of confidential user information stored on mobile devices, such as personal credentials, coordinates to financial assets, location data and photos. However, adversaries continuously devise unconventional ways to circumvent these security measures and exfiltrate sensitive data using novel covert channels.

Several works aim to establish covert channels in network communications by manipulating the properties of packets, such as timing and protocols [1], [2]. Other covert channels leverage measurable energy forms emitted by an electronic device as a medium for data exfiltration, such as electric power consumption [3], [4], [5], light [6], [7], temperature [8], acoustic [9], [10], [11], [12], and ElectroMagnetic (EM) emissions [13], [14], [15]. This type of covert channel is subtle and difficult to detect, since it relies on media that are not supposed to be used to carry information, making it suitable for data exfiltration from air-gapped systems [16]. In addition, the exfiltrated data are typically transmitted by modulating the intensity or timing of the energy form considered. Although most of the energy-based covert channels have focused on PCs [1], smartphone use also produces energy emissions, which can be used to establish a covert channel. Furthermore, Android is the most popular among mobile operating systems, with a market share of more than 70% in Q1 of 2024 [17].

In this paper, we propose a covert channel that achieves data exfiltration by modulating the power consumption of Android mobile devices during charging via a USB cable. On the transmitter side, our app encodes the data bits of sensitive information into intense resource usage with precise timing, which, in turn, influences the amount of electric current drawn from the USB power source. On the receiver side, we measure the electric current variations through a measuring tool and retrieve the original sensitive information using a robust decoder of our design.

Contributions. This paper, which is an extended and improved version of our previous work in [3], makes the following contributions:

- We present CovertPower, a power consumption-based covert channel that allows one to stealthily exfiltrate sensitive data from Android-based mobile devices.
- We design and implement a transmitting app that modulates bits into resource workload (i.e., CPU and writing on memory), resulting in an increase in power consumption.
- We propose a stealthy and inexpensive measurement tool tailored to measure the power traces produced by our covert channel.
- We develop a robust signal processing-based decoder that can automatically detect bit encoding, bitrate, and start/end of a transmission.

Received 18 September 2023; revised 11 March 2025; accepted 24 March 2025. Date of publication 31 March 2025; date of current version 4 September 2025. The work of Riccardo Spolaor was supported in part by the Research Fund for International Young Scientists (RFIS-I) of the National Natural Science Foundation of China under Grant 62350410480, in part by the Shandong University Future Young Scholars Project, and in part by the Natural Science Foundation of Shandong Province under Grant ZR2024MF149. The work of Veelasha Moonsamy was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the German Excellence Strategy - EXC 2092 CASA under Grant 390781972. *(Corresponding author: Mauro Conti.)*

Riccardo Spolaor, Yi Xu, and Xiuzhen Cheng are with the School of Computer Science and Technology, Shandong University, Qingdao 250100, China (e-mail: rspolaor@sdu.edu.cn; yxu@mail.sdu.edu.cn; xzcheng@sdu.edu.cn).

Veelasha Moonsamy is with the Horst Görtz Institute for IT Security, Ruhr University Bochum, 44801 Bochum, Germany (e-mail: email@veelasha.org).

Mauro Conti is with the SPRITZ Security Group, University of Padua, 35129 Padua, Italy, also with the Delft University of Technology (TU Delft), 2628, CD Delft, Netherlands, and also with the University of Washington, Seattle, WA 1410 USA (e-mail: mauro.conti@unipd.it).

Digital Object Identifier 10.1109/TDSC.2025.3556468

- We evaluate the attack with a thorough set of real-world experiments on 14 mobile devices (i.e., ten smartphones and four tablets) from seven different brands using wired and wireless charging technologies. Furthermore, we assess the performance of our attack using four metrics, varying parameters (i.e., bit encoding, burst generation, sampling rate) and the conditions of the mobile device, such as screen state, ongoing daily activities, and active wireless connection. In this evaluation, we demonstrate that our attack can reach a bitrate of up to 10 bps with a normalized Levenshtein distance lower than 0.08, and Indel and Jaro similarities higher than 0.9 in most experimental cases.

Difference with Preliminary Version. The novel contributions of this work, compared to the preliminary version in [3], lie in: (i) a complete redesign and implementation of the transmitter app to bypass the increasingly restrictive policies of recent Android releases; (ii) a reliable decoder that allows up to ten times faster bitrate; (iii) a cost effective solution for power consumption measurement; and (iv) a widely extended experimental evaluation under a more realistic threat model, including screen-on and -off activities, active wireless connections, and charging technologies.

Organization. The remainder of the paper is organized as follows. Section II reviews the related work. Section III provides background knowledge on Android power management and bit encoding techniques. Section IV describes the covert channel attack and its components. Section V reports the attack performance through an experimental evaluation. Section VI discusses possible implementations and limitations, and Section VII draws some conclusions.

II. Related Work

In this section, we review the existing work in the area of power channel in mobile devices and covert channels that rely on power signals.

A. Power Channel on Mobile Devices

In the literature, researchers have broadly investigated physical channel attacks on mobile and USB-powered devices [18], [19]. Among these attacks, a type of power channel attack consists of an adversary that measures the voltage and current drawn by a mobile device connected to a power supply. Cronin et al. in [20] discover a new security threat by exploiting the USB power line to infer user interactions on the touch screen. They develop a portable microcontroller-enabled power trace capture system using off-the-shelf hardware. Spolaor et al. in [21] develop a defensive mechanism based on the power channel to identify properties of USB-powered peripherals, among which type, model, and ongoing actions. Yang et al. in [22] develop a website fingerprinting attack based on power trace analysis. Lifshits et al. in [23] introduce a stealth attack vector that launches power channel attacks on mobile phones by replacing the original battery with a malicious one. Matovu et al. in [24] propose three power channel attacks that can be launched during the phone charging process: incoming call detection, and website and keystroke inference. Wang et al. in [25] propose an attack against voice assistants on the charging cable, which can inject and eavesdrop audio after modifying a shared powerbank.

Wireless charging technology has gained popularity, but has also introduced new security concerns. Wu et al. in [26] assess potential security and privacy vulnerabilities of Qi wireless charging technology. In particular, wireless device-to-charger communication can be non-intrusively interfered with and eavesdropped on. Zhang et al. [27] develop WirelessID, an authentication system that fingerprints wireless chargers by analyzing their signal while charging a device. La et al. in [28] demonstrate that the power channel of a wireless charger leaks enough information to achieve website fingerprinting. Liu et al. in [29] analyze the voltage dynamics to infer sensitive activities on a mobile device, such as passcode login, keystrokes, start an app, and speaker playback.

B. Covert Channels on PCs

Covert channels have been a powerful tool for conducting air-gapped attacks [16], allowing adversaries to extract sensitive information from isolated networks and transmit it to nearby receiving devices (attackers) through various media (optical, EM, vibration, acoustic, thermal, etc.). When a CPU is running, the PC generates heat, which can be exploited for covert communication. BitWhisper [8] leverages this phenomenon to bridge an air-gap between two PCs with temperature sensors. Air-gapped attacks can be carried out through power lines to leak sensitive data from closed network PCs. PowerHammer [5] uses the power line as a covert channel to exfiltrate data from air-gapped computers. It works by controlling the momentary utilization of the CPU core in the air-gapped PC to generate an electrical signal that transmits sensitive information to the power line. AIR-Jumper [6] proposes a covert channel between internal networks and remote attackers using surveillance cameras and infrared light, taking advantage of the fact that human eyes cannot see infrared light. LED-it-Go [7] uses hard-drive LED light as a means of fast data exfiltration under the condition that the receiver is within the line of sight.

The acoustic emissions produced by PCs while operating are also used to perform air-gapped attacks. Power-supplay [9] proposes a PC-to-PC covert channel based on the noise produced by the Power-Supply Unit (PSU) that can be received by a nearby microphone. Matyunin et al. in [10] utilize acoustic signals in the sub-bass and infrasonic range to establish a vibrational covert channel between speakers-equipped devices. Fansmitter [11] can exfiltrate sensitive information using the noise emitted by the CPU and PC fans. DiskFiltration [12] builds a covert channel that exfiltrates data from an air-gapped PC through acoustic signals generated by its HDD, operating at bit rates of 10,800 bits/hour. Manipulation of CPU workload on an air-gapped PC can generate EM emissions that enable data transmission to an external network. Matyunin et al. [30] use smartphone magnetic sensors to establish a covert channel between a laptop and a mobile device without additional equipment, firmware modifications, or privileged access to either of the devices. ODINI [13] exfiltrates data modulated as EM emissions produced by specific

TABLE I
COMPARISON OF COVERT CHANNEL ATTACKS RELYING ON PHYSICAL CHANNELS

Covert channel	Physical channel	Source device	Signal generation	Speed	Bitrate (bps)
BitWhisper [8]	Thermal radiations	PC	GPU and CPU	Very slow	0.002
PowerHammer [5]	Power consumption	PC	CPU	Fast	10
LED-it-Go [7]	LED Light	PC	Hard drive LED	Very fast	3k
AIR-Jumper [6]	Infrared radiations	PC	Video	Fast	20
Power-supplay [9]	PSU acoustic signals	PC	CPU	Very fast	50
Fansmitter [11]	Fans noise	PC	PC fans	Very slow	0.25
DiskFiltration [12]	Mechanical noise	PC	HDD I/O	Slow	3
Matyunin et al. [30]	Electromagnetic radiations	PC	I/O	Very slow	0.067
ODINI [13]	Electromagnetic radiations	PC	CPU	Very fast	40
MAGNETO [31]	Electromagnetic radiations	PC	CPU	Medium	5
MagView++ [14]	Electromagnetic radiations	PC	Video	Fast	8.9
Matyunin et al. [10]	Speaker vibration	PC	Speaker	Medium	3.5
Deshotels [32]	Ultrasounds	Smartphone	Ultrasound	Very fast	345
ShoutIMEI [33]	Ultrasounds	Smartphone	Ultrasound	Very fast	50
NFCT [3] (Previous work)	Power consumption	Smartphone	CPU	Slow	1.25
CovertPower (Current work)	Power consumption	Mobile devices	CPU and Memory	Fast	10

CPU operations. Similarly, MAGNETO [31] uses EM signals to exfiltrate data from air-gapped PCs to nearby mobile phones with EM sensors. Instead of intentionally adjusting CPU utilization, MagView++ [14] induces CPU workload by playing video to establish a covert channel based on EM emissions.

C. Covert Channels on Mobile Devices

Most previous covert channel research has focused on transmitting confidential data from closed network PCs. However, smartphones have become an indispensable part of daily life. Deshotels et al. in [32] propose a covert channel based on ultrasounds emitted from built-in speakers. Using a similar concept, ShoutIMEI [33] aims to exfiltrate the IMEI number from a victim smartphone in five feet.

Table I summarizes the covert channel attack in the literature. Most previous work exfiltrates data from victim devices by artificially manipulating the CPU workload. However, frequent CPU operations may increase the risk of detection by the operating system. Therefore, in this work, we investigate both CPU and memory operations. With a bitrate of 10 bps, our attack achieves a transmission speed comparable to or faster than most of the covert channels in the literature.

III. BACKGROUND KNOWLEDGE

In this section, we recall several concepts about the service and power management of the Android operating system in Section III-A and the bit encoding methods in Section III-B.

A. Android Services and Battery Optimizations

The Android OS has constantly been evolving its power management features to extend the up-time of a device between charges by optimizing battery usage. Android apps often rely on services (i.e., threads running in the background) to perform long-running tasks, significantly contributing to the overall device's power consumption. In what follows, we summarize the features of battery optimization and their impact on apps and services across the various Android versions.

As a default approach, the operating system briefly pauses apps after the device is idle (i.e., the screen is off). However, such an approach interferes with services that need to constantly work in the background (e.g., checking for new messages, social media notifications, and audio playback service). To ensure correct and timely work of these services, Android allows services to keep the CPU running by obtaining the `WAKE_LOCK` [34], However, this requires a permission classified as normal [35]. Subsequently, Android 6 implemented the Doze feature, which forces a device to enter a low-power state when idle (e.g., screen off). In particular, background services are not allowed to run when Doze is active, even if a service has wake-lock permission. Android 7 (Nougat) further improves such a feature with Doze-on-the-go, which uses motion sensors to infer when a device is in motion (e.g., in the user's pocket).

As a further restriction on background services, Android 8 (Oreo) limits their running time through the Background Execution Limits feature. However, foreground services are exempt from this restriction upon mandatory user notification in the notification bar. Subsequently, Android 9 (Pie) implemented the Adaptive Battery and App Standby Buckets features [36]. The former feature leverages machine learning to profile the user's behavior and allows only the background services she is likely to use. The latter feature categorizes apps by their usage patterns and assigns them a priority in terms of a quota of battery usage. In contrast, Android 10 (Quince Tart) only provides minor battery optimization, such as re-designing the system updates and introducing the dark theme (i.e., reduced light emitted by the screen). As an additional measure to optimize battery usage, Android 11 introduced the Flexible Power Controls feature. This feature allows users to customize the battery optimization of specific apps and services. Furthermore, Android 12 applies additional restrictions for launching foreground service while Android 13 alerts the user in the case of an excessive battery drain by an app over the last 24 hours. The most recent Android 14 requires developers to declare the type of a foreground service to match its prerequisites (e.g., a location service type needs `ACCESS_FINE_LOCATION` permission). The features mentioned above are released by the Android Open Source Project (AOSP), and they may vary according to the customized Android

versions (i.e., *Android forks*) pre-installed by a manufacturer (e.g., ColorOS by Oppo, One UI by Samsung).

When the device is in charging mode (as in our threat model), the Doze, App Standby Buckets, and Adaptive Battery features are disabled. In addition, the Background Execution Limits feature is active when the device is charging, causing a service to be killed shortly after entering idle mode unless it is a foreground service. Therefore, from Android 8 onward, the effectiveness of our previous attack in [3], [4] has been undermined, as it relies on a background service. As a solution, our current attack utilizes a foreground service which, together with our assumption of charging mode, allows our attack to bypass the above restrictions.

B. Bit Transmission

In this section, we provide some basic knowledge and definitions related to the modulation schemes used by digital communication systems to transmit digital information over a medium. In particular, we focus on the schemes suitable for achieving the covert channel proposed in this paper.

In general, digital modulation schemes represent binary data by varying one or a combination of three properties of an analog carrier signal: amplitude, phase, and frequency. Among such schemes, Amplitude-Shift Keying (ASK) transmits bits as variations in the amplitude of a carrier signal. A basic version of ASK is the *On-Off Keying* (OOK) where the presence or absence of a pulse corresponds to a one or a zero. The *period* is the time T_b it takes for a signal to complete an on and off cycle, thus transmitting a bit. Furthermore, a *duty cycle* d is the rate between the duration of the pulse ($p_t \leq T_b$) and the period T_b. For example, if a pulse lasts for $T_b/4$, its duty cycle is $d = 0.25$.

Given a communication channel, a line code defines the pattern of states that are considered to transmit digital data. To implement OOK, unipolar encoding is a line code that encodes bits as positive and neutral states in a transmission channel, e.g., positive and zero voltage, respectively. Such an encoding is suitable for our channel, since we can induce a pulse and control its duration. In particular, a *unipolar Return-to-Zero* (RZ) encoding represents a binary one with the presence of a positive voltage pulse with a duty cycle $d < 1$, after which the value returns to zero for $T_b(d-1)$. In contrast, the absence of a pulse (zero voltage) for a period T_b represents a binary zero. In a *unipolar Non-Return-to-Zero* (NRZ) encoding, the pulse has a duty cycle $d = 1$. In the case of consecutive ones, the signal remains in a positive voltage without returning to a zero voltage.

Manchester coding is a particular case of *Binary Phase-Shift Keying* (BPSK), which encodes binary data as transitions between positive and neutral states in the carrier signal. In particular, the Manchester code employs pulses with $d = 0.5$ and ensures a transition for each bit independently of its value, which is useful for recovering the signal's clock. Typically, positive-to-neutral and neutral-to-positive state transitions represent binary ones and zeros, respectively. Instead, the *Differential Manchester* encodes binary data in the polarity of the transition from the previous bit b_{t-1}. While for a zero the signal maintains the same transition used to transmit the previous bit, for a one the signal inverts the transition's polarity (i.e., positive-to-neutral of b_{t-1} becomes neutral-to-positive for b_t). Differential Manchester encoding is less error-prone in noisy channels and helps recover the signal clock.

IV. Our Covert Channel Through Power Line

In this section, we describe our attack that exfiltrates the target information from a mobile device by establishing a covert channel based on power consumption. For an overview, our proposed covert channel involves the two main components depicted in Fig. 1: a user's mobile device as the sender (on the left side) and a USB power supply monitored by an adversary as the receiver (on the right side). To recharge the mobile device, the user connects it to the USB power supply. The user has previously installed our CovertPower app on such a mobile device. The CovertPower app starts to transmit the target information by encoding it into power bursts with a specific timing. In the adversarial USB power supplier, the measuring tool monitors these power bursts and decodes them to retrieve the original target information.

In what follows, we first define the threat model, assumptions, and use cases in Section IV-A. Then we describe in detail the components of our covert channel: transmitting via the CovertPower app in Section IV-B, and receiving through the adversarial USB power supply in Section IV-C.

A. Threat Model and Use Cases

The purpose of our work is to investigate the security implications of establishing a covert channel using the power line between a mobile device and a power supply to which it is connected. Our method adheres to the covert channel paradigm, which involves two communicating parties: a transmitter (the CovertPower app) and a receiver (adversarial power supply). In what follows, we describe our threat model and state the assumptions for the two parties in the covert communication.

1) CovertPower App: On the transmitter side, we assume the CovertPower app can run as a foreground service on the battery-enabled Android device of the user. Such an app could be disguised as a standalone full-fledged app, included as a third-party library (e.g., advertisement, graphic engine) on an original app on the official Play Store, or a repackaged app on an unofficial app store. For simplicity, we henceforth refer to it as "app" even if a foreground service is enough to enable our covert channel. To enable our covert channel, the app only requires `FOREGROUND_SERVICE` permission. It is worth noticing that this permission is considered normal [35], i.e., it is only declared in the manifest and is granted without explicit user consent/interaction. However, the CovertPower app may require permission to access the *target information* to be exfiltrated. For example, if the target information is the device's contact list, the app needs to be granted the related permission (*target's permission*). Apart from the permission to access the target information, the CovertPower app does not need any dangerous permission (i.e., permissions that are granted only with the explicit consent of the user).

Security researchers have proposed methods to detect data exfiltration, among which network monitoring and information

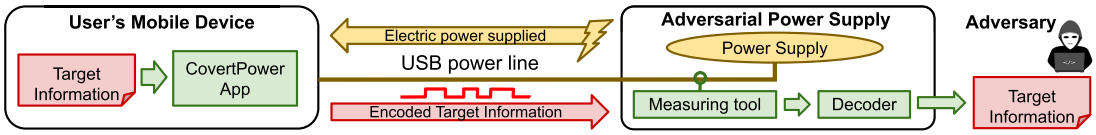


Fig. 1. Overview of the components involved in the covert channel.

flow tracking techniques. A Network-based Intrusion Detection System (NIDS) can monitor the network traffic of mobile devices to identify information theft attempts or connections to suspicious destination IP addresses. However, a NIDS may not identify a leak of sensitive information if it has been encrypted/obfuscated prior to its exfiltration through the monitored network. From an information flow perspective, taint analysis [37], [38] can track the usage of sensitive information and its derivatives (even if obfuscated or encrypted using a function or API) within an Android app, to identify possible leakage via network communications or forwarded to other unauthorized apps. Nonetheless, the CovertPower app can bypass these detection mechanisms since (i) it does not require wireless connectivity (e.g., Bluetooth, WiFi, 5 G), thus our attack is feasible even if a mobile device is set in airplane mode; and (ii) even if tainted, sensitive information appears to remain within the app, thus avoiding raising suspicion or alert. Differently from ordinary malicious apps that mainly exfiltrate data via a network connection, data exfiltration using our covert channel remains viable and undetectable even when network connections are monitored (e.g., NIDS), restricted, or unavailable (e.g., in an airplane, air-gapped networks).

We also assume that the target information and the transmission parameter are hard-coded or set autonomously (i.e., without external input) by the CovertPower app according to the device characteristics. As a realistic example, the CovertPower app can be disguised as a custom alarm clock, multimedia player, or e-book reader app. It is reasonable that these apps would require running a foreground service and access to internal storage in the case that the target information is a stored file.

2) Adversarial Power Supply: On the receiver side, we consider a realistic scenario in which a user has to recharge the battery of her mobile device via a USB power supply, which is under the control of an adversary. Such a power supply can be a charging station or a USB wall socket in a public facility (e.g., airport, train station, shopping mall, hotel), a USB power adapter, or a rental power bank. The adversary deploys a tool to monitor the electric current provided to the connected mobile devices. Such a monitoring tool is hidden inside the USB power supply, which appears ordinary even after a visual inspection. Moreover, we assume that the measuring tool is stealthy and passive so that it does not significantly alter the original output voltage and current from the USB power supply (i.e., does not affect the charging speed). For convenience, the measuring tool should be small and implemented with low-cost hardware. We propose an implementation of such a measurement tool in Section V-A. The power traces measured by the adversarial power supply could be stored locally and subsequently physically retrieved, or streamed via a private wireless network (e.g., WiFi, cellular) set up by the adversary. We assume that the adversary can easily infer the transmission parameters by analyzing the collected power traces. We do not rely in any way on the USB data line, as the user can use a power-only cable, have *Charging only* mode set by default on the device, or use a blocking device (e.g., USB condom [39]).

3) Use Case Scenarios: We present two realistic use case scenarios for our covert channel.

The first scenario considers a user whose activity is under surveillance and her device is monitored (e.g., taint analysis, network traffic monitoring). The user relies on our covert channel to secretly communicate with another recipient party evading such surveillance measures. The user knows that the recipient party monitors the power delivered by a power supply located in a public environment (e.g., train station, airport, shopping mall). Due to restrictive policies on the installation of apps on enterprise devices, the user installs the CovertPower app on her personal or disposable device to exfiltrate the sensitive information necessary to achieve more dangerous attacks. For example, the user could exfiltrate intelligence to carry out industrial or military espionage by covertly providing the IP address and credentials to access a protected network, revealing the instructions for a backdoor, or GPS coordinates of a secret facility's location.

The second scenario considers that a user is unaware of the presence of the CovertPower app installed on her device and the adversarial power supply. In this case, the user recharges her mobile device by connecting it to an adversarial power supply. Possible real-world combinations for apps (running CovertPower) and monitored power supply to enable the covert channel could be: (i) free charging stations in a shopping mall for customers who have the mall's app on their devices; (ii) a museum offering a free app for self-guided tours and free-to-use powerbanks; and (iii) a rental car company monitors the power outlets in their cars while requiring users to install its app for insurance and assistance purposes. This enables our covert communication for the exfiltration of the target information from the user's device that an adversary can use to carry out targeted attacks. For example, information on the device's build (e.g., OS version, API level, security patch level) or the list of installed apps to profile the user or identify specific OS/apps' vulnerabilities to exploit. As another example, the adversary could exfiltrate the contact list or files (e.g., password file, bank coordinates).

B. CovertPower App: The Transmitter

On the transmitting side, we design an Android app that manipulated the device's power consumption to generate a signal that carries the target information to be exfiltrated.

We define the elements involved in the transmission protocol for our covert channel:

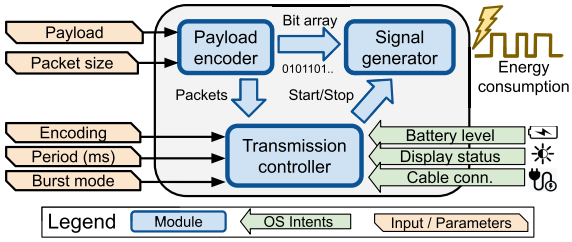


Fig. 2. The modules involved in the CovertPower app.

- *Payload* is the target information to be exfiltrated, which is encoded as a raw sequence of bits.
- *Syncword* is a characteristic sequence of bits that allows the receiver to synchronize. We use such a syncword in the *preamble* (at the beginning) and *postamble* (at the end) of a payload. In the case of a noisy channel, we may insert a syncword within the payload (as a midamble) to recover the synchronization in case of an error.
- *Packet* is the whole sequence of bits transmitted, which includes the preamble, payload, and postamble. We can split the payload into multiple packets if its length is excessively long.

In Fig. 2, we illustrate the three modules that compose the CovertPower app: *Payload encoder*, *Signal generator*, and *Transmission controller*.

1) Payload Encoder: This module takes the payload as input and outputs a bitstream. In general, the payload can be any information that can be serialized into an array of bits. Taking as an example a string of text as a payload, we first decompose the payload into an array of characters. We then convert each character into bits using its Base64 encoding in the American Standard for Information Interchange (ASCII). Therefore, the payload encoder produces the array of bits to be transmitted. In particular, this module can split long payloads into multiple packets by adding in the bitstream the proper syncwords (e.g., preamble, postamble) and, optionally, encryption or error-correcting codes (see Section VI-B) In our transmission protocol, each packet begins and ends with a preamble and a postamble, respectively. In particular, we choose the syncwords `0xAA` as preamble (i.e., "10101010" in binary) and `0xAB` as postamble (i.e., "10101001" in binary), since they are different and easily identifiable.

2) Signal Generator: Converts each bit in the bitstream into the related bursts of power consumption. These bursts will generate a signal that can be measured by the tool within the adversarial power supplier (wired and wireless chargers). This module takes as input the parameters for signal generation, which are the bit *encoding* (i.e., NRZ, Manchester, Differential Manchester), the bit *period* (i.e., bits per second), and the *burst method*. To obtain these bursts of power consumption, this module can use a power-consuming resource of the mobile device such as CPU, screen, flashlight, and file. To achieve a stealthy transmission, we cannot use resources, such as the screen and flashlight, that can be easily seen and raise suspicion by a user. Therefore, we propose two CPU-based and three File-based *burst methods* to generate power consumption.

Algorithm 1: Signal Generation Via a Burst Method.

```
Input: bitstream, method, period
for bit in bitstream do
    if bit = 1 then
        // bit = 1
        end ← GetCurrentTime() + period
        while GetCurrentTime() ≤ end do
            burst(method)
    else
        // bit = 0
        Sleep(period)
```

CPU-based burst methods increase the CPU workload by continuously performing computationally demanding operations in a loop.

- *Float-point operations* (*Float*) effectively put the CPU under stress, since they cannot be optimized by the ART JIT compiler, as they run an undetermined number of times. In particular, we increment a floating-point variable by 0.001.
- *Matrix operations on a bitmap* (*Pixel*) can also stress the CPU, thus increasing power consumption. First, we generate a random bitmap with a specific height and width. Then, we rotate it with a random angle α through a rotation matrix M is given by

$$M = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{1}$$

File-based burst methods increase power consumption by performing operations on files in a loop. Since some customized versions of Android may detect intense CPU activity, these file-based methods are an alternative solution as an effective workaround. We implement three variants for generating a burst for a bit:

- *Open-Write-Close (OWC).* For each iteration, we open, write, and close a file.
- *Write-Write (WW).* We only perform write operations on a file during the whole loop.
- *Write-Flush (WF).* We force a flush after each write on a file at each iteration.

It should be noted that these operations do not require additional permissions (e.g., external storage) since the files involved are in the memory dedicated to our CovertPower app.

We use the aforementioned methods for the transmission of a bitstream according to Algorithm 1. For temporizing the bursts, we employ a check on the system clock down to milliseconds. For the "bottom" part of the signal, we use the *sleep* function to pause the current thread for a predefined time (in milliseconds).

3) Transmission Controller: This module coordinates the transmission, monitors the status of the device, and determines whether it is feasible to transmit through the covert channel. The transmission controller determines the parameters for the output of the signal generator module.

This module also aims to identify the proper conditions to maximize transmission success while achieving stealthiness, i.e., minimizing the chances of being discovered by a user. In particular, it checks whether certain conditions are satisfied. As a first condition, a USB charging cable must be connected, which can be checked via a broadcast receiver listening to `Intent.ACTION_BATTERY_CHANGED` and the value for `BatteryManager.EXTRA_PLUGGED`. As a second condition, the battery level should be high enough for the variations in power consumption to be measurable (see Section V-B). We monitor the battery level by obtaining the value of `BatteryManager.EXTRA_LEVEL` from the above intent.

The Android Debug Bridge (ADB) allows monitoring of the CPU workload and other resources of a connected Android device, thus a user may notice a regularity in CPU bursts disclosing an ongoing transmission via our covert channel. To guarantee a stealthy transmission, as a third condition, we verify that the ADB is not active/disabled by monitoring the value `Settings.Global.ADB_ENABLED`.

When in use, the screen of a mobile device is an energy-hungry resource, yet its power consumption is quite stable. Therefore, we can clearly observe our transmitting signal when the device's screen is on. However, an on-off screen transition while transmitting a signal can cause a "step" in power consumption, which can cause bit errors during decoding. For this reason, this module monitors changes (transitions) in the screen state by checking the value of `PowerManager.isScreenOn()` or using a broadcast receiver set to `Intent.ACTION_SCREEN_ON`. It should be noted that to monitor these conditions, CovertPower app does not need any additional permission. If our app receives a broadcast intent from the Android OS that invalidates one of the aforementioned conditions, the *transmission controller* module will interrupt the transmission. This module also keeps track of the packets that have already been successfully transmitted. Therefore, if a packet transmission has been interrupted, the transmission controller will mark this packet for retransmission.

Once the proper conditions are satisfied and the data are ready to be sent, the app starts a foreground service where the signal generator runs as a thread. Using a foreground service, we can run such a thread in the background and reduce the risk of being killed by the Android OS. However, certain customized versions of Android may need additional actions to avoid restrictions due to battery optimization measures (see Section VI-D).

C. Adversarial Power Supply: The Receiver

In this section, we describe the receiver of our covert channel, a USB power supply controlled by the adversary. The adversarial power supply employs a measuring tool to passively collect power traces from the connected mobile device. To achieve a practical attack on a realistic scenario, we consider an easily conceivable measuring tool, which relies on small-sized and low-cost hardware (see Section V-A).

In Fig. 3, we show how to retrieve the target information transmitted from the power traces. In what follows, we describe the three sequential steps involved in this processing and the related challenges: *Signal processing*, *Bitstream decoding*, and *Payload finder*.

Fig. 3. Processing steps for decoding the target information.

1) Signal Processing: We consider the power traces collected as a signal that encodes digital information. However, we need to apply signal processing techniques since power traces can contain (i) noise due to the channel and acquisition method (hardware used by the measuring tool); (ii) interference by processes running concurrently to CovertPower app on the mobile device; (iii) a strong Direct Current (DC) component due to the normal power consumption of the connected device; and (iv) a variable sampling rate due to the measuring tool.

We report an example of a signal after each processing step in Fig. 4. First, we address the problem of a variable sampling rate. We assess whether this problem occurs by relying on the UNIX timestamp provided by the measuring tool for each sample. Therefore, we apply a linear interpolation of the signal by adjusting the sampling frequency. Second, we standardize the signal to remove its DC component. In particular, we subtract the mean of every sample from the signal. However, since the baseline can vary over time due to the Android power management system, we apply standardization using a wide sliding window (i.e., 10 seconds). Third, we filter out high-frequency noise and interference by applying a low-pass filter to the signal. We use a Butterworth filter because it has a flat response in the passband region, making it suitable for filtering square signals. To avoid filtering useful frequencies for our encoded bitstream, we set the cutting frequency at the maximum bitrate $br_{max} = 1/min(T_b)$.

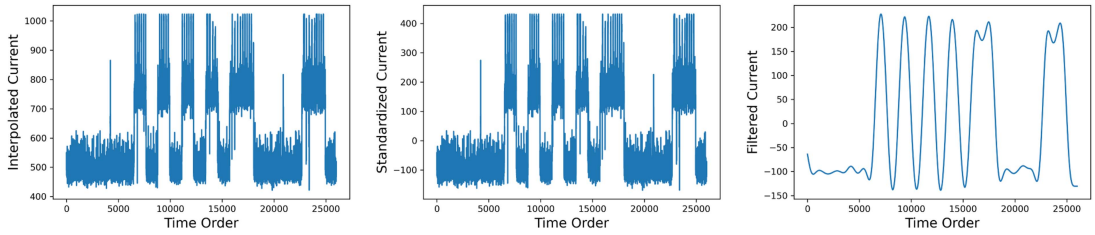
2) Bitstream Decoding: This step aims to identify the bit period and recover the bit stream from the processed signal. Since we transmit a data packet using an OOK modulation, the bitstream decoder extracts from the signal the high-low and low-high transitions and their duration by considering a threshold set at the zero of the processed signal.

Despite being aware of the possible values for the bitrate (e.g., from 1 to 10 bps), we assume that the receiver may not know the actual bitrate $\hat{b}$ in advance; therefore, the receiver infers it from the signal. To achieve this task, we rely on the self-synchronization property for a Manchester-based encoding. However, using NRZ encoding we may observe a high or low state for a prolonged time due to sequences of multiple consecutive ones or zeros, even after applying a measure to obtain a zero-mean signal. Therefore, we need to apply the following strategy to infer $\hat{b}$ for the NRZ encoding, which is also effective for a Manchester-based encoding.

We consider a portion of the signal with a fixed number of transitions Tr. Given a transition, we define the duration d_i as the time difference (in seconds) between a transition tr_i and the next transition tr_{i+1}, where the signal state (high or low) remains constant. We obtain $\widehat{br}$ by solving the following equation:

$$\widehat{br} = \underset{br \in [1,\ldots,10]}{\operatorname{argmin}} \left(\sum_{i=0}^{|Tr|-1} \lfloor d_i \cdot br \rfloor \cdot \left(d_i \bmod \frac{1}{br} \right) \right)$$

In other terms, our aim is to find the best value for b that minimizes the overall differences between the real and expected



(a) Interpolated signal. (b) Standardized signal. (c) Filtered signal.

Fig. 4. Steps of signal processing for the current traces.

bit duration (or multiple consecutive bits) for a bitrate br among the considered ones. However, this process might identify more than one bitrate under rare circumstances, i.e., br_1 is a multiple of br_2 and the signal contains multiple Tr bits with the same value. In this case, we can repeat this process on a new signal sample until we obtain an individual $\widehat{br}$.

Once we determine the data bitrate, we convert the sequence of transitions of the signal into a bitstream. Starting from a transition tr_i at time t_i, we advance a cursor on the signal's time domain until we either encounter another transition tr_{i+1}. Hence, we take the time $d_i = t_{i+1} - t_i$ and verify that $d_i \approx 1/(\widehat{br} \pm 10\%)$ otherwise we mark tr_{i+1} as erroneous. The current bit is one if the signal stays above the threshold after the transition tr_i, otherwise the bit is zero (i.e., below the threshold). However, since NRZ encoding allows, we find the number x of consecutive zeros or ones by solving $d_i \approx x/(\widehat{br} \pm 10\%)$.

3) Payload Finder: Upon successful recovery of the bits, we aim to retrieve the payload. However, we cannot know in advance the position of a packet. Therefore, the payload finder uses Algorithm 2 to inspect the bitstream and identify the syncwords for the preamble (begin) and postamble (end) of a packet. Such an algorithm has to deal with errors that may occur in a syncword.

Initially, the payload finder identifies and counts the occurrences of preambles and postambles within the bitstream (lines 1–2). If no preamble occurs, the algorithm considers as a payload the $payloadsize$ bits preceding a postamble (lines 3–10). Similarly, if no postable occurs, the algorithm considers as a payload the $payloadsize$ bits following a preamble (lines 11–15). If multiple preambles and postambles occur (some may be erroneous), the algorithm considers their Cartesian product to extract the possible payload (lines 16–19). We then examine such payloads to identify the ones with $payloadsize$ bits. If no preamble or postamble occurs, we rerun Algorithm 2, giving as input a partial preamble and postamble.

Algorithm 2: Find Payloads.

```
   Input: bitstream, preamble, postamble, payloadsize
   Output: payloads (set)
   // find the matches in the bitstream
1  preambles ← bitstream.matches(preamble)
2  postambles ← bitstream.matches(postamble)
   // No preamble found
3  if preambles.count = 0 then
4      p ← bitstream
5      for post in postambles do
6          start ← p[post.index - payloadsize]
7          end ← p[post.index]
8          if start≥ 0 then
9              payloads.add(p[start:end])
10             p ← p[end:]
   // No postamble found
11 else if preambles.count = 0 then
12     p ← bitstream
13     for pre in preambles do
14         p ← p[pre.index + preamble.length]
15         payloads.add(p[:payloadsize])
   // Preamble and postamble found
16 starts ← preambles.indexes
17 ends ← postambles.indexes
   // Cartesian product between starts and ends
18 for (s,e) in starts×ends do
19     payloads.add(bitstream[s+preamble.length:e])
```

V. Experimental Evaluation

In this section, we first describe the setup of our experiments and the transmission parameters. We then experimentally evaluate and discuss the performance of our attack in different settings, activities, and charging technologies.

A. Experimental Setup

We provide the details for the software and hardware tools, mobile devices, and performance metric used in the experimental evaluation.

1) Hardware Implementation: To measure the electrical current supplied to the connected mobile device, we implement the tool based on low-cost and small-size hardware as shown in Fig. 5. By splicing a current sensor onto the GND wire of a USB extension cable, we do not require any hardware modification on either the USB power supply or the connected device. In a wired charging scenario, we employ two sensors in series:

- *Shunt*. 0.1Ω resistor to measure the current as the voltage drop, which we amplify with an AD623AN instrumentation amplifier.
- *Hall*. ACS723 low-current sensor that measures current using the Hall effect.

In a wireless charging scenario, we use another current sensor as we describe in Section V-H. We acquire the sensor data with the Arduino Nano microcontroller's built-in Analog-to-Digital Converter (ADC). The Arduino code reads the ADC data at a

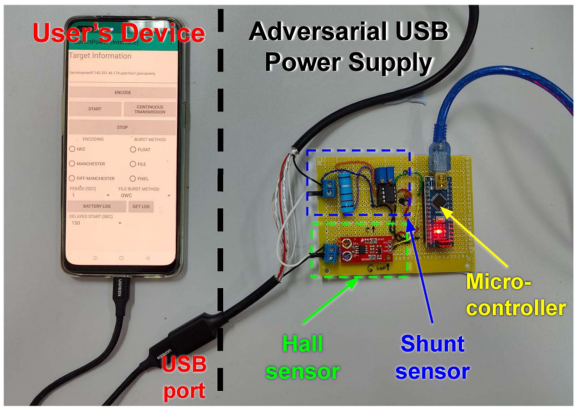
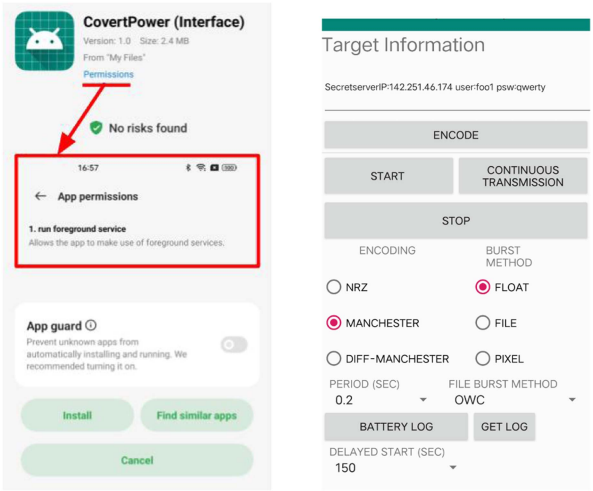


Fig. 5. Measuring tool for wired charging.



(a) Installation. (b) App interface.

Fig. 6. Screenshots for the prototype of CovertPower app.

constant sampling rate (up to 1 kHz) and transmits them to the decoder through serial communication (set at 115200 bps). As a possible alternative implementation, such ADC data could be stored on an SD card or wirelessly streamed with related Arduino modules (e.g., ESP8266). It should be noted that such a small tool can easily be concealed in a power supply, powerbank, or charging station. The total cost of the measuring tool is less than 30USD.

2) Software Implementation: We implement the software for both the transmitter and the receiver sides. On the transmitter side, we develop our CovertPower app using Android Studio Arctic Fox and Deveco Studio 3.0 for Android and HarmonyOS, respectively. Fig. 6 shows two screenshots for the CovertPower app prototype. During installation, the operating system displays the information in Fig. 6(a). As we can notice, CovertPower app only requests foreground service permission, which is considered a normal permission. We use the interface in Fig. 6(b) to select transmission parameters, such as encoding, bitrate, burst method, and delayed start. We use the VLC Player app v3.5.4 to play audio/video contents.

On the receiver side, we develop a script to process the data transmitted by the measuring tool. We implement such a script using the Python v3.10 and Scipy v1.10.1 libraries for signal processing. We connect the measuring tool and decode the signal on a Dell workstation with an Intel Core i5-7500 @ 3.40 GHz 16 GB RAM running Windows 11.

3) Experimental Devices: We evaluate our attack on ten smartphones and four tablets from seven major manufacturers, the details of which are reported in Table II. We update every device to the latest version available for its stock operating system. To replicate a realistic usage scenario, we do not remove any app installed by default, modify the original settings, or stop any app running in the background by default. We install CovertPower app allowing it to run background processes on battery optimization mechanisms (more details in Section VI-D). For each device model, we use the stock fast-charging USB power supply.

4) Performance Evaluation: To evaluate the performance of our covert channel attack, we transmit packets with a payload of size 512 bits to attain a sufficiently long bit-sequence for evaluation purposes without an excessively long transmission duration (below 9 minutes at 1 bps). In addition, we empirically assess that the size of the payload has a negligible influence on the performance of our attack. A payload contains a random sequence of letters and numbers of ASCII code. It is worth mentioning that we do not apply error detection or correction techniques. Upon decoding, we compare the sent and received payloads.

We assess the quality of transmission using three edit distance-based and one signal-quality metrics.

Levenshtein distance [40] computes the minimum number of edits (i.e., insertions, deletions, and substitutions) that are required to transform a bit sequence S_1 into another one S_2. We normalize it [41] dividing it by $max(|S_1|, |S_2|)$ where $|\cdot|$ is the length of a sequence.

Indel Similarity measures the normalized similarity (in a range between 0 to 1) between two sequences S_1 and S_2 as

$$Indel = 1 - \frac{Ind(S_1, S_2)}{|S_1| + |S_2|},$$

where $Ind()$ is minimum number of indels (i.e., insertion-deletion) required to turn S_1 into S_2.

Jaro Similarity [42] computes the normalized similarity between two sequences as

$$Jaro = \frac{1}{3}\left(\frac{m}{|S_1|} + \frac{m}{|S_2|} + \frac{m - (tr/2)}{m}\right) \quad \text{for } m > 0,$$

where m is the number of matching elements and tr is the number of transpositions (i.e., matching elements that are not in the right order).

Signal-to-Noise Ratio (SNR) assesses the quality of received signal in terms of signal power in decibel (dB) as

$$SNR = 10 \cdot \log_{10} \frac{S_t^2}{N^2} = 10 \cdot \log_{10} \frac{S_t^2}{(S_t - S_r)^2}$$

TABLE II
EXPERIMENTAL DEVICES, OPERATING SYSTEMS, AND CHARGER INFORMATION

ID	Brand	Model	Type	Year	Display	OS Version	OS Distribution	Fast charge technology	Charger	Wireless
OpZ1	Oppo	Reno 4 Z1	Phone	2020	6.4	Android 13.0	ColorOS 13	Oppo SUPERVOOC	65W	✗
Hn7	Huawei	Nova 7 SE	Phone	2020	6.5	HarmonyOS 3.0.0	HarmonyOS 3.0.0	Huawei SuperCharge	40W	✗
H10x	Huawei	Honor X10	Phone	2020	6.6	HarmonyOS 3.0.0	HarmonyOS 3.0.0	Huawei SuperCharge	23W	✗
Op8	Oppo	Oneplus 8	Phone	2020	6.5	Android 13.0	OxygenOS 13	Warp Charge 30T	30W	✗
Op12	Oppo	Oneplus 12	Phone	2023	6.8	Android 14.0	ColorOS 14.0	Oppo SUPERVOOC	100W	✓
Sx1	Sony	Xperia 1 IV	Phone	2022	6.5	Android 14.0	Xperia 64.2.F	Sony Quick Charge	30W	✓
Gf4	Samsung	GZ Flip 4	Phone	2022	6.7	Android 14.0	One UI 6.1	Samsung Quick Charge	25W	✓
Hm9	Huawei	Honor Magic 6	Phone	2024	6.8	Android 14.0	MagicOS 8.0	Huawei SuperCharge	66W	✓
Mrzr	Motorola	Razr 40 Ultra	Phone	2023	6.9	Android 13.0	MyUX 6.0	-	33W	✓
Dge	Doogee	V20 pro	Phone	2023	6.4	Android 12.0	Doogee EEA	-	30W	✗
OTab	Oppo	Pad Air2	Tablet	2023	11.4	Android 14.0	ColorOS 14	-	33W	✗
HTab	Huawei	MatePad Pro	Tablet	2019	10.8	HarmonyOS 3.0.0	HarmonyOS 3.0.0	Huawei Quick Charge	20W	✓
STab	Samsung	GTab S6 Lite	Tablet	2020	10.4	Android 13.0	One UI 5.1	-	8W	✗
XTab	Xiaomi	Pad 6S Pro	Tablet	2024	12.4	Android 14.0	HyperOS 1.0.10	QuickCharge 4.0	120W	✗

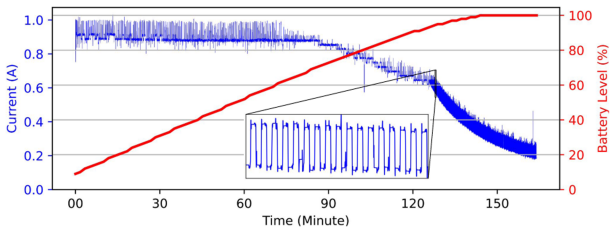
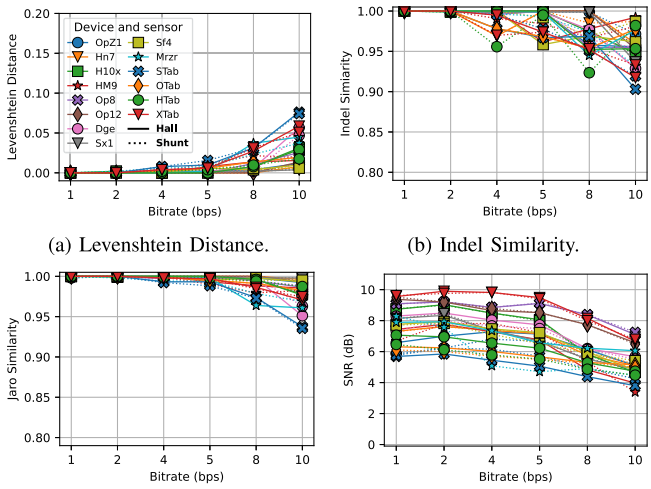


Fig. 7. Battery level (in red) and current drawn (in blue) over time for Op8.



(a) Levenshtein Distance. (b) Indel Similarity.

(c) Jaro Similarity. (d) Signal-to-Noise Ratio.

Fig. 8. Baseline performance of our covert channel attack under different bitrate and current sensors.

where the noise N is the difference between a signal transmitted S_t by the app and the signal received S_r measured at the power source.

B. Study on the Impact of Battery Level

In the charging state, the USB power supply provides energy to all components of a connected device, including the CPU, memory, screen, and Li-ion battery [43]. Once the other components are powered, most of the excess available power serves to recharge the battery, which overshadows the actual consumption of the other components when observing a power trace. However, the amount of current absorbed by the battery decreases with increasing battery level until it reaches a minimum when the battery is fully charged.

We conduct this preliminary experiment to study how battery level affects the observation of the power consumption of other components; thus, the one produced by the CovertPower app signal. In this study, we collect power traces while recharging *Op8*, in which CovertPower app continuously transmits zeros and ones. We throttle the maximum current output of the power supply to 900 mA. Fig. 7 shows the current draw (blue line) and the level of the battery (red line) over time. At a low battery charge level, most of the current supplied goes to charge the battery, which is known as the constant voltage stage. We can only observe the signal transmitted by CovertPower app as timed current variations (signal in the magnifying rectangle) when the battery level is above 90%. These results are consistent with similar studies related to power channel analysis [22], [25], [28]. Therefore, CovertPower app would not transmit unless the battery is above such a level.

C. Baseline Experiment

To establish a baseline for the performance of our covert channel, we collect the current traces from the devices in Table II while CovertPower app transmits packets using *NRZ* encoding and *pixel* burst generation method, with the device's screen turned and airplane mode enabled (disabling WiFi and Bluetooth). We collect current traces from both the *hall* and *shunt* sensors at a sampling rate of 1 kHz.

We report the results for the baseline in Fig. 8 varying the transmission bitrate. We can observe that our covert channel Levenshtein distance remains below 0.04 for all devices at 8 bps, while it is lower than 0.06 at 10 bps for most devices (except *STab*). Considering the shunt sensor, the received information is very similar to the transmitted one, since for all devices Indel and Jaro similarities are above 0.91 and 0.94, respectively, even at the maximum bitrate considered. In terms of signal quality, the SNR is generally stable up to 5 bps and has an attenuation of around -2 dB at 10 bps. Note that restrictions on the launch of foreground services introduced up to Android 14 (see Section III-A) do not affect CovertPower app functionalities nor the quality of the transmitted signal.

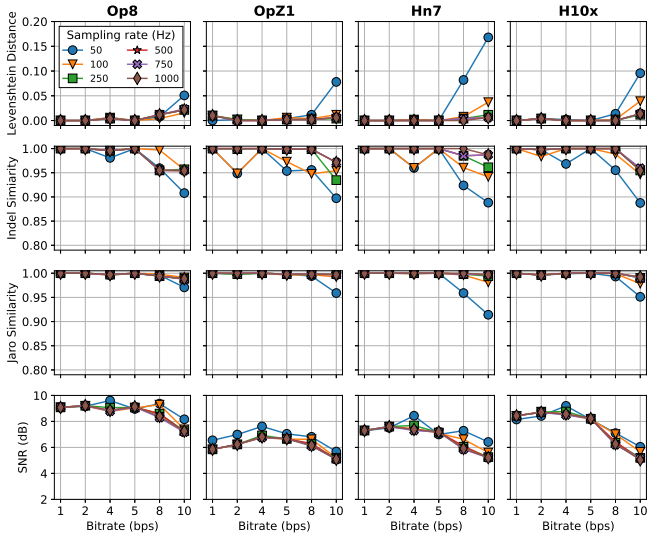


Fig. 9. Performance under different sampling frequency.

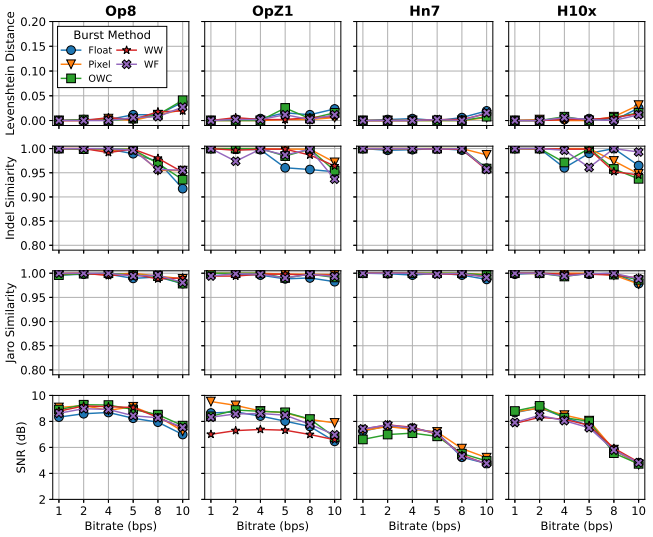


Fig. 10. Performance under different burst generation methods (CPU-based and File-based).

We can also observe a negligible performance difference between *hall* and *shunt* sensors. However, the *shunt* sensor is preferable to *hall* sensor since it is less expensive and easier to conceal. Henceforth, we consider power traces from *shunt* sensor and select a representative subset of devices: *H10x*, *Op8*, *Hn7*, and *OpZ1*.

D. Impact of Sampling Rate

We evaluate the robustness of our decoder by varying the sampling rate of the collected current traces from 50 Hz to 1 kHz. In this experiment, we transmit data under the baseline settings in Section V-C (i.e., airplane mode on, screen off, *NRZ*, *pixel*). In Fig. 9, we report the results for the individual devices.

Overall, a sampling rate equal to or above 250 Hz does not significantly affect the edit distance and signal quality. When the sampling rate is 100 Hz and 50 Hz, the Levenshtein distance for *Hn7* suddenly increases to 0.08 and 0.16 (with a consequent decrease in Indel and Jaro similarities), respectively, making transmission difficult to recover in the latter case. Therefore, our decoder can still recover data transmitted with a high bitrate even from sensor data with a sampling rate as low as 100 Hz.

E. Impact of Burst Generation Method

In this experiment, we evaluate the robustness of our covert channel by comparing different methods for the burst generation. In particular, CovertPower app uses the two CPU-based (i.e., *float* and *pixel*) and three file-based methods (i.e., *OWC*, *WW*, and *WF*). While varying the burst generation method, we apply the other baseline settings in Section V-C (i.e., airplane mode on, screen off, NRZ, 1 kHz sampling rate).

The results in Fig. 10 show that the generation method does not significantly affect the performance of our attack. Nonetheless, *pixel* and *WW* offer better overall performance, as its Levenshtein distance remains below 0.04 and Indel similarity and Jaro similarity are above 0.94 and 0.97, respectively, even under the most unfavorable conditions (i.e., on *H10x* and *OpZ1* at 10 bps). In addition, signal quality follows a similar trend as the baseline, with only a notable exception for *WW* in *OpZ1* where the SNR is attenuated yet stable at around 7 dB. Therefore, the file-based *WW* method is a viable alternative to avoid occasional alerts about apps that use excessive CPU.

F. Impact of Encoding Methods

We study the performance considering different encoding methods: *NRZ*, Manchester (*Man*), and Differential Manchester (*D.Man*). As we explain in Section III-B, Manchester-based encodings have $d = 0.5$, while *NRZ* has a duty cycle $d = 1$. In other words, the pulse duration of a Manchester-based encoding at 5 bps is equivalent to that of *NRZ* at 10 bps. Therefore, we consider a range of bitrates from 1 to 5 bps to ensure a fair comparison between the encoding methods. Similarly to the above experiments, we apply the baseline settings in Section V-C (i.e., airplane mode on, screen off, *pixel*, 1 kHz sampling rate).

As reported by the results in Fig. 11, we achieve good performance in terms of low Levenshtein distance and high Jaro similarity for all three bit encoding methods. However, the Indel similarity is relatively lower than the Jaro similarity for *Man* and *D.Man*, which indicates that sequence errors can be solved with substitutions (i.e., a 0 mistaken for a 1 or vice versa) rather than with insertion or deletion. We can also observe that Manchester-based encodings have a lower SNR than *NRZ* for three of the four devices considered. Although they are less stable than *NRZ*, the Indel similarity of Manchester-based encodings does not fall below 0.9 at 5 bps.

G. Impact of Activities on Mobile Devices

Since a user can use her device while charging, we assess the robustness of our covert channel with ongoing activies, distinguishing between screen off and screen on activities. In the

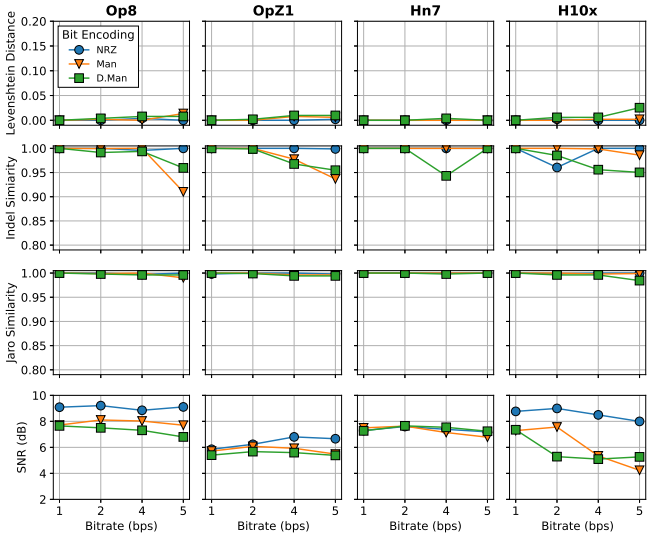


Fig. 11. Performance under different signal encoding.

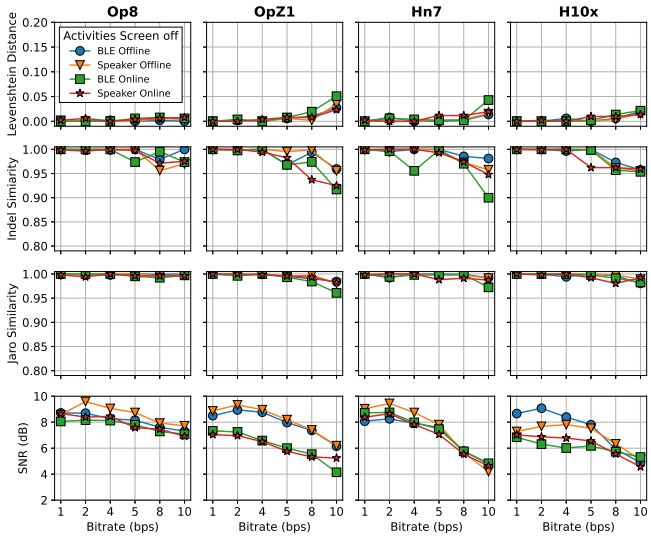


Fig. 12. Performance with ongoing activities while the screen is *off*.

following experiments, we use *NRZ* encoding, a 1 kHz sampling rate, and enable networking (i.e., airplane mode off).

1) Activities With the Screen Off: Unlike the experiments above where the device is idle, we consider a user listening to audio (e.g., music, podcast) while her device is charging with the screen off. As a source of audio data, we use a local file in the device's storage (*Offline*) and audio streaming over a WiFi network connection (*Online*). As audio output, we consider the built-in *Speaker* and Bluetooth Low Energy-connected headphones (*BLE*), with volume set at 60%.

As shown by results in Fig. 12, audio playback in the *offline* settings does not significantly affect the performance as Indel similarity stays above 0.95 independently from the audio output *speaker* and *BLE*. However, in the *online* settings, we generally achieve lower performance than their *offline* counterparts, since audio streaming through an active WiFi connection negatively

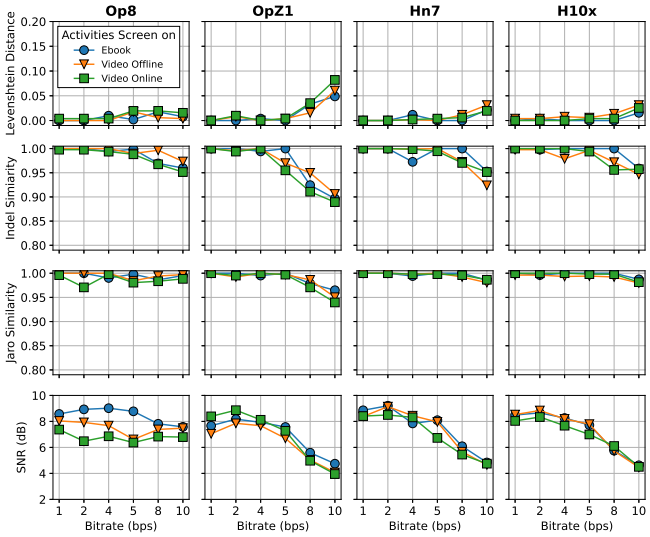


Fig. 13. Performance with ongoing activities while the screen is *on*.

affects covert transmission. To further confirm this insight, we can observe that the SNR of *online* instances is attenuated due to the noise introduced by an active WiFi connection. In summary, our attack overall achieves good performance in terms of low Levenshtein distance (at most 0.05); high Jaro similarity (above 0.95); and a mostly high Indel similarity (above 0.95), except for the *online* instances of *OpZ1* and *Hn7* at 8 and 10 bps.

2) Activities With the Screen on: A user can use her charging device for activities that require the screen to be on. Therefore, we consider three representative activities: reading a book (*Ebook*), watching a *Video* from a stored file (*Offline*) or in streaming (*Online*). We set the brightness of the screen to 50% and, for *Video*, use the built-in speaker as an audio output with volume at 60%.

In Fig. 13, we report the performance of our attack while the screen is on and activity is ongoing. The power consumption of the active screen is generally stable and does not significantly affect the performance of our attack. Increasing bitrate, the SNR for *screen on* activities has a similar attenuation to the *online* settings for *screen off* activities. We achieve a comparable performance with *screen off* activities for most of the activities and devices considered. However, as expected, *Video Online* activity has slightly lower performance than *Video Offline* and *Ebook* due to moderate interference from an active wireless connection. Similarly with previous experimental instances, the transmission on *OpZ1* at 8 and 10 bps has the lowest Indel (within 0.91-0.95 and around 0.9, respectively) and Jaro similarities (around 0.97 and 0.95, respectively); and, consistently, the highest Levenshtein distance (up to 3.5 and 8.2, respectively).

H. Impact of Charging Technology

As an alternative to the traditional USB cable, wireless charging has been an emerging technology to recharge the battery of mobile devices. In this experiment, we evaluate the performance of our attack in a wireless charging scenario.

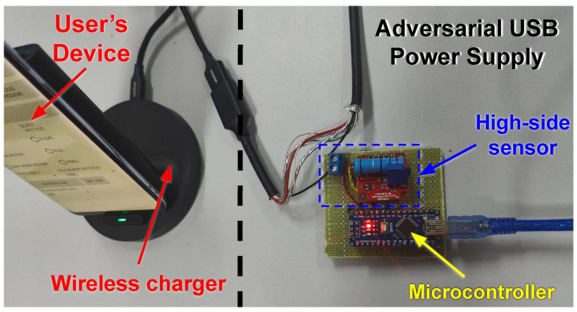


Fig. 14. Measuring tool for wireless charging.

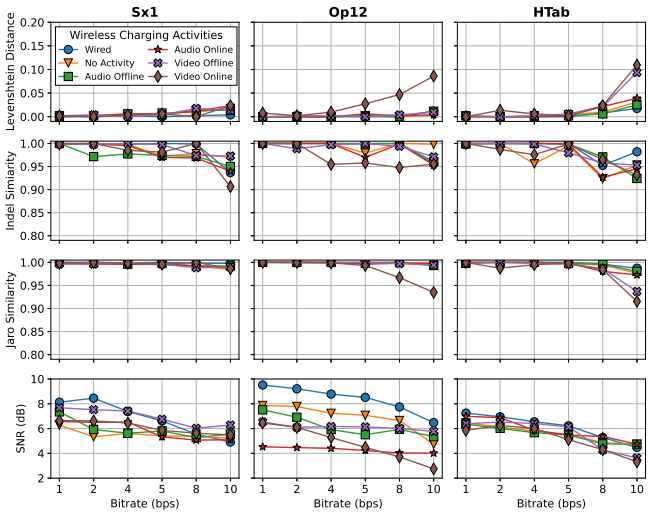


Fig. 15. Performance with ongoing activities under wireless charging.

1) Measuring Tool Setup: As wireless charging pad, we use a Xiaomi Vertical Wireless Charger (Universal Fast Charge 20 W) powered by a USB cable from a power supplier (5 V, 2 A max). We splice the VCC wire of the USB cable with a INA219 High-Side DC current sensor and connect this sensor to an Arduino Nano microcontroller via I2C interface. This microcontroller can acquire sensor data at a maximum sampling rate of 100 Hz. Fig. 14 illustrates the setup for wireless charging, which costs less than 15USD.

2) Evaluation: In this evaluation, we consider the three devices that enable wireless charging: two smartphones (*Hp30*, *Mate*) and one tablet (*HTab*). Consistent with the baseline settings, CovertPower app produces bursts using *pixel* and *NRZ* encoding. We make sure that the device is placed on the wireless charger pad so that the coil-to-device alignment and angle are consistent across experiments. For direct comparison, we collect power traces with a device in idle mode (i.e., screen off, airplane mode on) for *Wired* and wireless charging (*No Activity*). Moreover, we select four activities among the ones in Section V-G, two with screen off (i.e., *Audio Offline* and *Online* using the speaker as the audio output), and two with screen on (i.e., *Video Offline* and *Online*).

Fig. 15 shows the results for our covert channel in wireless charging settings. We can notice that in most cases idle wireless charging (*No Activity*) has a slightly lower performance than wired charging, due to the noise introduced by wireless charging technology. On the one hand, in the presence of activities in a wireless charging scenario, the Levenshtein distance increases with increasing bitrate, especially for the two video-related activities in *HTab* (at most 0.11 at 10 bps) and *Video Online* in *Op12* (0.05 and around 0.09 at 8 and 10 bps, respectively). On the other hand, SNR, Indel and Jaro similarities gradually decrease when increasing the bitrate, with a noticeable exception for the Indel similarity of *Video Online* in *Op12* (stable around 0.95 at 4 bps and above), indicating that the errors that occurred are due to misinterpreted bits rather than insertions or deletions. Therefore, we demonstrate that our attack is also effective in most activities in a wireless charging scenario.

VI. Discussion and Limitation

In this section, we discuss the implementation choices and limitations of our covert channel attack.

A. Encoding Selection

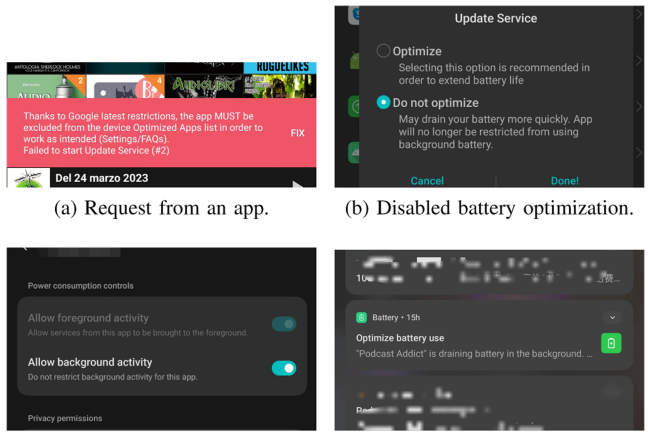
In the previous version of this attack [3], we considered unipolar RZ as bit encoding. This encoding has a duty cycle $d = 0.5$ with a pulse of duration $T_b/2$ to represent a one and no pulse for a zero (see Section III-B). The rationale for adopting *RZ* encoding was to minimize pulse duration (thus CPU burst) to avoid raising CPU temperature and excessively prolonging charging time, which may alert the user. However, these concerns do not hold anymore due to recent technological advancements of mobile devices in terms of battery and multi-core CPUs. We experimentally assess that our attack is stealthy even when using *NRZ* and running for an extended time (see study in Section V-B) since it does not increase the temperature or prolong the charging time. Using *NRZ* encoding and other improvements to the decoder, we achieve an almost tenfold increase in transmission speed compared to [3].

B. Error Characterization and Recovery

In our experiments, we observed a time delay in the "reaction" of the power signal as a result of burst-to-idle and idle-to-burst transitions. Despite its limited entity, at high bitrates, this reaction delay could negatively impact the identification of transitions by the bitstream decoding process (see Section IV-C2). In particular, the power signal can remain above or below the zero threshold for a time less than $1/2T_b$, which also affects the quality of the signal in terms of SNR. Thus, the bitstream decoding process may fail to identify two adjacent transitions as in the case of isolated burst or idle periods. Therefore, the main source of errors is caused by a bit-flip (i.e., substitutions). As evidence of this, we can notice in Fig. 8 that the performance in terms of Indel similarity is worse than the Jaro similarity and Levenshtein distances. This is because, differently from the other two edit-distance metrics, the Indel similarity does not consider substitution errors.

Due to the isolated nature of these errors, a possible solution to detect them is to use one parity bit for each bit word of

length D bits. Upon detection of an error within a word, further analysis of the signal may reveal the missing transition that caused the bit to flip. However, this solution would not hold in the case of multiple errors. Alternatively, mainly designed for multiple error detection, Cyclic Redundancy Check (CRC) methods compute for each bit word a fixed-length checksum (e.g., 8 bits for CRC-8). However, since its error detection ability depends on the checksum length, the likelihood of undetected errors increases with the length of the bit word. As a more suitable solution, Hamming codes can detect multiple errors and correct one error at the cost of additional P parity bits spread in a word of T bits ($D = T - P$ bits of the actual payload), i.e., Hamming(T, D). However, such codes introduce a non-negligible overhead of P/D bits, thus resulting in a longer transmission time. In terms of numbers, transmitting a payload of 512 bits applying Hamming(12,8) allows to correct at most 64 b errors (at most one error per bit word) at the cost of 128 additional bits to be transmitted (i.e., an overhead of around 33.3%).



(a) Request from an app. (b) Disabled battery optimization.

(c) Allowing background services. (d) Battery draining alert.

Fig. 16. Example of requests made by a legitimate media player app in order to work properly and alert.

C. Device Specific Attack

From our experiments, we assess the attack performance generally decreases with increasing bitrate and that the device considered plays a role in this, especially for high bitrates. Compared to other devices, this decrease is evident on *STab* and *XTab* devices (see Section V-C). Note that these devices out-of-the-box equip ROMs that are customized by the manufacturers (i.e., One UI, HyperOS, respectively). which contain a high number of pre-installed system processes running in the background, which likely hinder our covert signal at the highest bitrate considered. To cope with this inconvenient, CovertPower app could adjust the maximum bitrate for exfiltration according to the device, obtaining this additional information through `android.os.Build`. On the receiver side, our robust decoder can automatically identify the bitrate by observing the preamble.

D. Avoiding Battery Optimization Mechanisms

Android has become release after release increasingly strict against background processes and with battery optimization (see Section III-A). Unfortunately, such mechanisms may also prevent legitimate apps from working correctly. As a representative example, we show the effects of the restrictions of Android 14 (and previous) on a popular podcast player app in Fig. 16. First, the app requests to disable battery optimization to allow the update service to search for new episodes (Fig. 16(a) and (b)). Second, to avoid interruption of audio playback, the user must manually disable the restrictions on background activity in the app properties (Fig. 16(c)). However, the system notifies the user with an alert about excessive battery usage by the app (Fig. 16(d)). It is worth specifying that CovertPower app has never raised such an alert, not even after its long runtime for the experiment in Section V-B

We show that an app can ask the user to disable this restriction, explaining that it is necessary to allow the app to work correctly. Therefore, an adversary can disguise CovertPower app as an app that is likely to rely on the foreground services, such as a media player, a fitness tracker, or an alarm clock. Alternatively, CovertPower app can also be embedded in a foreground service of another popular app by repackaging and distributing it on a third-party market. As a foreground service needs a mandatory notification on the status bar, the above examples of disguise apps can issue notifications without raising any suspicion (e.g., playback controls, tip of the day, incoming alarm). Moreover, we assess that disabling/hiding app notifications does not affect the related foreground services from running correctly.

E. Countermeasures

Over the years, researchers have also investigated methods to counteract private information leakage through side channels. In most cases, the authors of an attack also propose possible software or hardware countermeasures against such an attack.

On the one hand, some works aim to obfuscate power leakages by perturbing the power consumption of the device. Liu et al. in [29] also design an app that aims to randomize the voltage to reduce the risks of privacy leakage in a wireless charging scenario. In a wired scenario, Matovu et al. in [24] propose to evaluate two defense methods, a hardware-based and a software-based solution. These methods randomly perturb the current drawn during charging, thus masking the unique patterns of user activity. Unfortunately, such countermeasures introduce overhead in terms of increased charging time or resource consumption on the device (e.g., CPU and memory).

On the other hand, Cronin et al. in [20] suggest that the leakage channel could be stopped by inserting a low-pass filter into the charging circuit of the device, which can remove the informative high-frequency component of the signal. However, this countermeasure would not be effective against our attack, since its transmitting signal has a low frequency (below 20 Hz). Following this approach, we could mitigate our attack using a low-pass filter with a cut-off frequency of at most 1 Hz, which would basically allow only for the DC component. For example,

such a filter can be implemented using several operational amplifiers (e.g., TI TS321) connected in a daisy-chain or a passive RL filter. However, since such a filter cannot be ideal, it would only attenuate our signal because of the small difference in magnitude between DC (i.e., 0 Hz) and the cut-off frequency.

VII. Conclusion

In this paper, we presented CovertPower, a covert channel attack for data exfiltration from Android devices. Our covert channel relies on timed variations in a device's workload, which results in a corresponding electric current drawn that can be measured at the power supply. On the one hand, we designed the transmitter app that induces such a workload while coping with the restrictions of recent Android releases. On the other hand, our robust decoder makes our attack effective independently of the screen state and the ongoing activities in a device.

We carried out a comprehensive experimental evaluation including 14 mobile devices and tested different attack variants (e.g., burst method, bit encoding) and settings (e.g., ongoing activities, wired and wireless charging). The experimental results confirm the reliability of our covert channel in the exfiltration of sensitive data, which would hinder the security and privacy of mobile users. For this reason, we also discuss the pros and cons of existing countermeasures that could mitigate such an attack.

Acknowledgment

We would like to thank Heyuan Shi, Riccardo Bonafede, Laila Abudahi, Łukasz Chmielewski, Elia Dal Santo and Ivan Martinovic for their support and insightful suggestions.

References

[1] J. Tian, G. Xiong, Z. Li, and G. Gou, "A survey of key technologies for constructing network covert channel," *Secur. Commun. Netw.*, vol. 2020, pp. 1–20, 2020.

[2] A. Gangwal, S. Singh, R. Spolaor, and A. Srivastava, "Blewhisperer: Exploiting BLE advertisements for data exfiltration," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2022, pp. 698–717.

[3] R. Spolaor, L. Abudahi, V. Moonsamy, M. Conti, and R. Poovendran, "No free charge theorem: A covert channel via USB charging cable on mobile devices," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2017, pp. 83–102.

[4] R. Spolaor, B. Riccardo, M. Veelasha, and M. Conti, "No free charge theorem 2.0: How to steal private information from a mobile device using a powerbank," in *Proc. Appl. Cryptogr. Netw. Secur.: 15th Int. Conf.*, 2018, pp. 83–102, https://www.blackhat.com/eu-18/briefings/schedule/index.html#no-free-charge-theorem-how-to-steal-private-information-from-a-mobile-device-using-a-powerbank-12630

[5] M. Guri, B. Zadov, D. Bykhovsky, and Y. Elovici, "Powerhammer: Exfiltrating data from air-gapped computers through power lines," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1879–1890, 2019.

[6] M. Guri and D. Bykhovsky, "Air-jumper: Covert air-gap exfiltration/infiltration via security cameras & infrared (IR)," *Comput. Secur.*, vol. 82, pp. 15–29, 2019.

[7] M. Guri, B. Zadov, and Y. Elovici, "Led-it-go: Leaking (a lot of) data from air-gapped computers via the (small) hard drive led," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, M. Polychronakis and M. Meier, eds. Berlin, Germany: Springer, 2017, pp. 161–184.

[8] M. Guri, M. Monitz, Y. Mirski, and Y. Elovici, "Bitwhisper: Covert signaling channel between air-gapped computers using thermal manipulations," in *Proc. IEEE Comput. Secur. Found. Symp.*, 2015, pp. 276–289.

[9] M. Guri, "Power-supplay: Leaking sensitive data from air-gapped, audio-gapped systems by turning the power supplies into speakers," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 313–330, Jan./Feb. 2021.

[10] N. Matyunin, Y. Wang, and S. Katzenbeisser, "Vibrational covert channels using low-frequency acoustic signals," in *Proc. ACM Workshop Inf. Hiding Multimedia Secur.*, 2019, pp. 31–36.

[11] M. Guri, Y. Solewicz, A. Daidakulov, and Y. Elovici, "Fansmitter: Acoustic data exfiltration from (speakerless) air-gapped computers," *Comput. Secur.*, vol. 91, 2020, Art. no. 101721.

[12] M. Guri, Y. Solewicz, A. Daidakulov, and Y. Elovici, "Acoustic data exfiltration from speakerless air-gapped computers via covert hard-drive noise ('diskfiltration')," in *Proc. Eur. Symp. Res. Comput. Secur.*, Springer, 2017, pp. 98–115.

[13] M. Guri, B. Zadov, and Y. Elovici, "ODINI: Escaping sensitive data from faraday-caged, air-gapped computers via magnetic fields," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1190–1203, 2019.

[14] X. Ji, J. Zhang, S. Zou, Y. Chen, G. Qu, and W. Xu, "MagView: Data exfiltration via CPU magnetic signals under video decoding," *IEEE Trans. Mobile Comput.*, vol. 23, no. 3, pp. 2486–2503, Mar. 2024.

[15] M. Guri, "Air-gap electromagnetic covert channel," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 4, pp. 2127–2144, Jul./Aug. 2024.

[16] J. Park, J. Yoo, J. Yu, J. Lee, and J. Song, "A survey on air-gap attacks: Fundamentals, transport means, attack scenarios and challenges," *Sensors*, vol. 23, no. 6, 2023, Art. no. 3215. [Online]. Available: https://www.mdpi.com/1424-8220/23/6/3215

[17] Statista, "Mobile operating systems' market share worldwide," 2024. [Online]. Available: https://www.statista.com/statistics/272698

[18] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard, "Systematic classification of side-channel attacks: A case study for mobile devices," *IEEE Commun. Surv. Tut.*, vol. 20, no. 1, pp. 465–488, Jan. 2017.

[19] H. Liu, R. Spolaor, F. Turrin, R. Bonafede, and M. Conti, "USB powered devices: A survey of side-channel threats and countermeasures," *High-Confidence Comput.*, vol. 1, no. 1, 2021, Art. no. 100007.

[20] P. Cronin, X. Gao, C. Yang, and H. Wang, "Charger-surfing: Exploiting a power line side-channel for smartphone information leakage," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 681–698.

[21] R. Spolaor, H. Liu, F. Turrin, M. Conti, and X. Cheng, "Plug and power: Fingerprinting USB powered peripherals via power side-channel," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2023, pp. 1–10.

[22] Q. Yang, P. Gasti, G. Zhou, A. Farajidavar, and K. S. Balagani, "On inferring browsing activity on smartphones via USB power analysis side-channel," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 5, pp. 1056–1066, May 2017.

[23] P. Lifshits et al., "Power to peep-all: Inference attacks by malicious batteries on mobile devices," in *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 4, pp. 141–158, 2018.

[24] R. Matovu, A. Serwadda, A. V. Bilbao, and I. Griswold-Steiner, "Defensive charging: Mitigating power side-channel attacks on charging smartphones," in *Proc. 10th ACM Conf. Data Appl. Secur. Privacy*, 2020, pp. 179–190.

[25] Y. Wang, H. Guo, and Q. Yan, "GhostTalk: Interactive attack on smartphone voice system through power line," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2022, pp. 1–15.

[26] Y. Wu, Z. Li, N. Van Nostrand, and J. Liu, "Time to rethink the design of QI standard? security and privacy vulnerability analysis of QI wireless charging," in *Proc. Annu. Comput. Secur. Appl. Conf.*, 2021, pp. 916–929.

[27] J. Zhang, Z. Wang, X. Ji, W. Xu, G. Qu, and M. Zhao, "Who is charging my phone? identifying wireless chargers via fingerprinting," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2992–2999, Feb. 2021.

[28] A. S. La Cour, K. K. Afridi, and G. E. Suh, "Wireless charging power side-channel attacks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 651–665.

[29] J. Liu et al., "Privacy leakage in wireless charging," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 2, pp. 501–514, Mar./Apr. 2024.

[30] N. Matyunin, J. Szefer, S. Biedermann, and S. Katzenbeisser, "Covert channels using mobile device's magnetic field sensors," in *Proc. 21st Asia South Pacific Des. Automat. Conf.*, 2016, pp. 525–532.

[31] M. Guri, "Magneto: Covert channel between air-gapped systems and nearby smartphones via cpu-generated magnetic fields," *Future Gener. Comput. Syst.*, vol. 115, pp. 115–125, 2021.

[32] L. Deshotels, "Inaudible sound as a covert channel in mobile devices," in *Proc. 8th USENIX Workshop Offensive Technol.*, 2014, Art. no. 16.

[33] K. Pandya, B. Borisaniya, and B. Buddhadev, "ShoutIMEI: Ultrasound covert channel-based attack in android," in *Proc. Secur. Privacy Data Analytics: Sel.*, 2022, pp. 293–301.

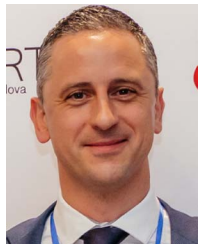
[34] Android Developers Reference, "Choose the right API to keep the device awake," Accessed: Feb. 28, 2025. [Online]. Available: https://developer.android.com/develop/background-work/background-tasks/awake

[35] Android Developers Reference. Android manifest permission, Accessed: Feb. 28, 2025. [Online]. Available: https://developer.android.com/reference/android/Manifest.permission/
[36] Android Developers Reference, "Optimize for doze and app standby," Accessed: Feb. 28, 2025. [Online]. Available: https://developer.android.com/training/monitoring-device-state/doze-standby
[37] W. Enck et al., "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, pp. 1–29, Jun. 2014.
[38] S. Arzt et al., "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *ACM Sigplan Notices*, vol. 49, no. 6, pp. 259–269, 2014.
[39] C. Parker and C. Parker, "Don't be a smartphone dummy," in *Firewalls Don't Stop Dragons: A Step-by-Step Guide to Computer Security andPrivacy for Non-Techies*. Apress, 2020, pp. 351–373.
[40] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Proc. Sov. Phys. Doklady*, 1966, vol. 10, no. 8, pp. 707–710.
[41] A. Marzal and E. Vidal, "Computation of normalized edit distance and applications," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 9, pp. 926–932, Sep. 1993.
[42] M. A. Jaro, "Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida," *J. Amer. Statist. Assoc.*, vol. 84, no. 406, pp. 414–420, 1989.
[43] G.-J. Chen and W.-H. Chung, "Evaluation of charging methods for lithium-ion batteries," *Electronics*, vol. 12, no. 19, 2023, Art. no. 4095.

Riccardo Spolaor (Member, IEEE) received the PhD degree in brain, mind, and computer science from the University of Padua, Italy, in 2018 and the MSc degree in computer science from the same university, in 2014. He is an associate professor with the School of Computer Science and Technology, Shandong University, Qingdao Campus, China, from 2019. He previously worked as a research associate with the University of Oxford, U.K., in 2018-19. His main research interests are privacy and security of mobile devices and Machine Learning-based physical channel analyzes to detect malware or infer private information. He has published more than 30 papers in top international conferences and journals, such as *IEEE Security & Privacy*, NDSS, IEEE INFOCOM, *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Mobile Computing*, etc. He has also served as a Program Committee member of several international conferences, including ACM WWW, ACM CCS, and ESORICS.

Yi Xu received the BSc degree in computer science and technology from the Guilin University of Electronic Technology, China, in 2021. She is currently working toward the MSc degree with the School of Computer Science and Technology, Shandong University, Qingdao campus, China, from 2021. Her research interests include IoT security, side channel analysis, data storage, and timely backup.

Veelasha Moonsamy received the PhD degree from Deakin University, in Melbourne (Australia). She is a tenured research faculty with the Horst Görtz Institute for IT Security and a PI within the Excellence Cluster CASA with Ruhr University Bochum in Germany. Her research interests revolve around security and privacy of mobile devices, in particular side- and covert-channel attacks, malware detection, and mitigation of information leaks at the application and hardware level.



Mauro Conti (Fellow, IEEE) received the PhD degree from the Sapienza University of Rome, Italy, in 2009. He is full professor with the University of Padua, Italy. He is also affiliated TU Delft, The Netherlands; and University of Washington, Seattle, USA. After his PhD, he was a post-doc with Vrije Universiteit Amsterdam, The Netherlands. In 2011, he joined as an assistant professor with the University of Padua, where he became an associate professor, in 2015, and a full professor in 2018. He has been a visiting researcher with GMU, UCLA, UCI, TU Darmstadt, UF, and FIU. He has been awarded a Marie Curie Fellowship (2012) by the European Commission and a Fellowship by the German DAAD (2013). His research is also funded by companies, including Cisco, Intel, and Huawei. In the area of Security and Privacy, he has published more than 500 papers in top-most international journals and conferences. He has been editor-in-chief of *IEEE Transactions on Information Forensics and Security*, area editor-in-chief for *IEEE Communications Surveys & Tutorials*, and associate editor for several journals, including *IEEE Communications Surveys & Tutorials*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Information Forensics and Security*, and *IEEE Transactions on Network and Service Management*. He was program chair for TRUST 2015, ICISS 2016, WiSec 2017, ACNS 2020, CANS 2021, CSS 2021, WiMob 2023 and ESORICS 2023, and general chair for SecureComm 2012, SACMAT 2013, NSS 2021 and ACNS 2022. He is a Fellow of the AAIA, Senior Member of the ACM, and Fellow of the Young Academy of Europe.

Xiuzhen Cheng (Fellow, IEEE) received the MS and PhD degrees in computer science from the University of Minnesota – Twin Cities, in 2000 and 2002, respectively. She is a professor in the school of computer science and technology, Shandong University. Her current research interests include wireless and mobile security, cyber physical systems, wireless and mobile computing, sensor networking, and algorithm design and analysis. She has served on the editorial boards of several technical journals and the technical program committees of various professional conferences/workshops. She also has chaired several international conferences. She worked as a program director for the US National Science Foundation (NSF) from April to October in 2006 (full time), and from 2008 to 2010 (part time). She received the NSF CAREER Award in 2004. She is a member of ACM.