When Side Channels Meet Covert Channels: A Practical Fully Sensor-Driven Attack Chain

Ye Wang[0009-0002-0450-8939], Yuying Li[0009-0008-1059-4806], Bo Luo[0000-0001-8196-2436], and Fengjun Li[0000-0003-4079-2228]

EECS/I2S, The University of Kansas, Lawrence, KS, USA
yeah_wong@ku.edu, yuyingli@ku.edu, bluo@ku.edu, fli@ku.edu

Abstract. Permission-free motion-sensor-based side and covert channels remain effective primitives for sensitive information extraction and exfiltration on Android devices. While prior work extensively studies these channels in isolation, modern defenses have made continuously operating attacks increasingly impractical. We observe that the missing component is a stealthy control plane that coordinates attack activation and communication. We propose a lightweight motion-sensor-only control protocol that orchestrates sensor-based side and covert channels into an event-driven attack chain. Using accelerometer-based eavesdropping and vibration-based covert communication as a case study, our design leverages natural phone vibrations as stealthy control signals without requiring sensitive APIs or protected resources. Our evaluation shows that the proposed attack chain reduces anomalous energy consumption from 51.5% to 5.2% while maintaining a 96.7% attack success rate, and successfully bypasses state-of-the-art static analysis and sensor-anomaly detection systems.

Keywords: Side Channel, Covert Channel, Attack Chain

1 Introduction

Android smartphones store and process large amounts of sensitive information, including authentication credentials, location traces, and financial data [24]. Although the Android security framework strictly restricts direct access to such data [22], prior work has shown that sensor-based side channels can bypass these protections and infer sensitive information from indirect physical signals [28]. For example, a wide range of smartphone sensors, including motion sensors, microphones, cameras, and magnetometers, have been exploited to perform keystroke inference attacks that recover sensitive inputs such as PINs and passwords [28].

Android primarily mitigates sensor misuse through permission-based access control mechanisms. However, unlike high-risk sensors such as cameras or microphones, motion sensors remain largely permission-free because they support ubiquitous functions such as screen rotation, gaming, and fitness tracking. This design decision creates a persistent attack surface [28]. Modern Android systems

Table 1. Existing accelerometer-based eavesdropping attacks: machine learning models, minimum supported sensor sampling rates, and corresponding attack accuracy.

Eavesdropping	Accelword [36]	AccelEve [5]	Vibphone [32]	inertiEAR [13]	AccEar [16]	iSPYu [37]
Publication Year	2015	2020	2021	2022	2022	2023
Min sampling rate (Hz)	No limitation	200	50	200	167	120
ML Model	Decision Tree	DenseNet	DenseNet	DenseNet	cGAN	BiLSTM/TF
Accuracy (Highest/200 Hz)	90%/ NA	73%/64%	94.0%/54.2%	NA/78.8%	4.8/6.6	61%/40%

Note: InertiEAR was specifically designed to perform eavesdropping under a 200 Hz sampling-rate limitation. AccEar evaluates reconstruction quality using Mel-Cepstral Distortion (MCD).

(since Android 12) limit motion sensor sampling rates to reduce the practicality of motion sensor-based side-channel attacks [15]. For instance, as illustrated in Table 1, without sampling-rate constraints, lightweight models such as decision trees can achieve satisfactory eavesdropping performance [36]. Under sampling-rate limitations, lightweight models become ineffective. To maintain acceptable attack accuracy under these restrictions, attackers increasingly rely on more complex architectures, such as DenseNet [13, 32], conditional GAN [16], Bidirectional Long Short-Term Memory (BiLSTM), and transformers [37].

However, abnormal energy consumption can be used to detect malicious or anomalous behaviors on mobile devices [9]. In particular, aggregate energy consumption (i.e., the sum over a given time window) [18] is a key time-domain feature used for detection. Our evaluation (Figure 4 in Section 5.2) shows that the aggregate energy consumption of continuously running ML models, especially the mid-size and large-size ones, is much higher than regular mobile app energy consumption. This indicates that continuously executing complex ML models on smartphones is relatively easy to detect. One possible strategy to reduce on-device energy consumption is to offload computation by transmitting raw sensor data to a remote server. However, transmitting raw sensor streams that contain sensitive information is increasingly being monitored [21].

Furthermore, even if sensitive information is successfully inferred locally, exfiltrating the data remains challenging. Conventional communication channels such as network sockets, Bluetooth, or NFC are either heavily monitored or require explicit permissions, making them unsuitable for stealthy transmission [12]. Physical covert channels based on motion sensors offer an alternative; however, they typically provide limited bandwidth and require nearby receivers, imposing strict requirements on coordinating when transmissions should occur [17].

In summary, existing defenses have significantly reduced the practicality of sensor-based attacks on the channel plane, where sensitive information is extracted or transmitted through side or covert channels. Continuously operating side-channel inference or covert communication often introduces noticeable overhead, increasing detectability. In contrast, the control plane, which governs attack activation and coordination, remains largely unexplored. Without an efficient control mechanism, sensor-based attacks cannot reliably support event-driven inference or stealthy covert communication.

In this paper, we show that restoring the practicality of sensor-based attacks requires only a lightweight control protocol in the control plane. Based on this insight, we design a fully sensor-driven control mechanism that orches-

trates side-channel inference and covert-channel transmission without invoking sensitive system APIs or conventional communication interfaces. Using motion sensor-based eavesdropping and vibration-based covert communication as a case study, we leverage natural phone vibrations as control signals and implement the entire control protocol solely using accelerometer data. Experimental evaluation demonstrates that the proposed attack chain significantly reduces abnormal energy consumption from 51.5% to 5.2% while maintaining a 96.7% attack success rate. Moreover, the resulting attack chain successfully evades automated Android security analysis tools and sensor-anomaly detection systems. Our main contributions are as follows:

- We present the first study of sensor-based side and covert channels from the control-plane perspective. We show that a lightweight control protocol operating solely in the control plane is sufficient to restore the practical feasibility of sensor-based attacks under modern Android constraints.
- We leverage natural phone vibrations as control-protocol signals, enabling control logic to operate entirely within the sensor-data layer and avoiding suspicious system-level interactions.
- We implement end-to-end prototypes to evaluate the effectiveness and stealth of our attack chain, and provide an in-depth analysis of the fundamental limitations of existing defenses.

2 Background

2.1 Motion Sensor–based Side Channel Attacks

Motion sensors capture motion-related features such as direction, velocity, and orientation [10]. In particular, accelerometers measure acceleration induced by physical displacement, which allows them to capture vibrations generated by user interactions and environmental stimuli [6]. Consequently, accelerometers have been widely exploited in side-channel attacks. Attackers develop malicious applications to collect accelerometer data and process it locally to extract sensitive information. For example, vibrations induced by keystrokes can be captured to recover sensitive inputs such as PINs and passwords [36].

Accelerometers can also reconstruct vibrations generated by sound, enabling the leakage of private speech content during phone conversations [16]. Recent research shows that speech signals emitted from smartphone speakers can be recovered through motion sensors [5]. A representative case study is recovering PIN digits from speech signals emitted during phone calls [5]. Compared with keystroke inference, which can be mitigated using randomized keyboard layouts [30], motion-sensor-based acoustic eavesdropping has attracted increasing attention because it can reveal longer and richer semantic information [28].

2.2 Motion Sensor–based Covert Channel Attacks

Conventional communication channels on smartphones are often monitored or protected by explicit permissions [19]. As a result, covert channels have been

explored as alternative mechanisms for stealthy data exfiltration [14]. One such approach leverages vibration motors as signal emitters and accelerometers as receivers. Vibrations generated by a device's onboard motor [2] or by a nearby smartphone placed on the same surface [17] can propagate through the shared medium and be detected by motion sensors.

These vibration-based channels can operate even when devices are locked or running in the background, making them attractive for covert communication. However, their bandwidth is extremely limited. Prior work reports transmission rates as low as 0.64 bits per second [17], which makes it impractical to transmit raw sensor data. Instead, such channels are typically used to transmit small amounts of abstracted sensitive information, such as PIN codes. In addition, vibration-based covert channels impose several physical constraints. They typically require close proximity between devices (e.g., within 1 meter) and depend on suitable surface materials to propagate vibration signals effectively [25].

2.3 Motion Sensor Security Mechanisms

Android adopts a permission-based access control framework to regulate applications' access to sensitive system resources, including sensors and hardware interfaces [22]. Applications must declare required permissions in their manifest files, while users grant access either during installation or at runtime for dangerous permissions. Sensors that directly expose sensitive information, such as cameras and microphones, are therefore tightly protected through explicit permission checks. In contrast, motion sensors such as accelerometers and gyroscopes remain largely permission-free because they support ubiquitous functionalities.

To mitigate motion sensor-based eavesdropping, Android introduced sampling rate restrictions starting from Android 12 (API level 31). Previously, applications could obtain the maximum hardware-supported sampling rate by using *registerListener* with the `SENSOR_DELAY_FASTEST` mode. However, now Android versions cap the sampling rate in this mode at 200 Hz by default [15]. Higher rates require `HIGH_SAMPLING_RATE_SENSORS` permission. In addition, the *SensorDirectChannel* interface further restricts the sampling rate to `SENSOR_DELAY_NORMAL`, which typically corresponds to approximately 50 Hz. This restriction aims to balance usability and privacy by limiting data granularity while preserving functionality for legitimate applications. Many common applications, such as gesture recognition and health monitoring, operate within the 100–200 Hz range, making it impractical to further reduce the sampling-rate cap.

Beyond platform-level restrictions, several research efforts aim to detect malicious sensor usage through anomaly detection. For example, 6thSense [27] performs context-aware intrusion detection by modeling expected sensor-state transitions of user activities using probabilistic and tree-based classifiers such as Markov models, Naive Bayes, and Logistic Model Trees. More recent approaches, such as SBTDDL [20], leverage deep learning models (e.g., LSTM) and augment training datasets with carefully crafted malicious behaviors that mimic benign sensor usage, improving robustness against evasive attacks.

3 The Problem and Threat Model

3.1 Challenges in Practical Sensor-Based Attacks

Challenges in Phone Call Inference. Sensor-driven Android applications are designed for lightweight, real-time processing, with short sensor buffers and algorithms optimized for low latency and energy consumption [15]. Motion sensors support "always-on" operation with minimal overhead, enabled by efficient signal-processing pipelines such as optimized FFT implementations [1, 33]. As a result, motion-sensor data is typically processed in a streaming manner rather than stored for offline analysis. This design also constrains sensor-based attacks. In particular, reconstructing speech-related information from low-resolution motion-sensor signals requires machine-learning models that operate directly on the incoming sensor stream. However, continuously running such inference modules incurs substantial computational and energy overhead, making the attack behavior anomalous and potentially detectable through resource-usage monitoring.

A natural strategy is to activate the expensive eavesdropping module only when a phone call is occurring. Yet obtaining call status through conventional system APIs requires sensitive permissions that weaken the stealth advantage of permission-free sensor attacks. Alternatively, inferring call status directly from raw motion-sensor data is challenging: under the sampling-rate limitations imposed by recent Android versions, vibration signals induced by speech are weak and easily obscured by environmental motion and background noise. Consequently, designing a lightweight and stealthy mechanism to infer call context and selectively trigger high-cost inference modules remains a key challenge for making sensor-based eavesdropping practical under modern Android defenses.

Challenges in Emitter Synchronization. Physical covert channels on mobile devices typically use hardware actuators (e.g., vibration motors or speakers) to transmit signals, while nearby receivers passively capture these signals through sensors. Because the emitter cannot reliably determine the receiver's presence or readiness in dynamic mobile environments, continuous broadcasting would be required to ensure successful transmission.

However, such continuous actuator-based signaling (e.g., repeated vibrations) is perceptible to users and therefore impractical for stealthy communication. Conventional short-range communication technologies (e.g., Bluetooth, Wi-Fi Direct, or NFC) cannot address this issue either, as they require explicit user authorization and system-level prompts that expose device coordination.

One possible alternative is to trigger vibrations through the Web vibration API. Although occasional vibration events are common in web applications [34], encoding sensitive data into vibration patterns requires the content to be delivered to the web layer, effectively exposing the transmitted information. Another approach is to send only trigger signals to a local application that invokes the vibration API, but this resembles a WebView privilege-bridging pattern in which web content indirectly accesses privileged system functionality [26], making it likely to be flagged by security analysis systems.

Table 2. Availability of call-connect haptic feedback across major Android vendors.

Brand	Call-connect haptic feedback	Example models (recent)
Samsung	Yes (explicit system setting)	Galaxy S22–S24, Galaxy A54–A56
OnePlus	Yes (in many OxygenOS builds)	OnePlus 9–12
Motorola	Sometimes	Moto G Power, Edge 40/Edge 50
Google Pixel	Off by default (available in Google Phone app settings)	Pixel 6–9
Xiaomi	Yes (commonly present in MIUI and HyperOS)	Xiaomi 12–14, Redmi K70/K80

These challenges motivate a fully sensor-driven attack chain. By exploiting vibration-induced artifacts in accelerometer readings, we design a lightweight control protocol that infers call initiation and termination without relying on call-state APIs and uses vibration signals to synchronize covert transmissions. This approach enables reliable control and data exfiltration using only permission-free motion sensors and the non-sensitive vibration permission, minimizing both resource overhead and the observable attack surface.

3.2 Threat Model

We target Android smartphones running Android 12 or later, which introduced sampling-rate limitations as a key security update. As of January 2026, Android 12 and later account for approximately 77% of worldwide Android market share [31], a proportion that continues to grow.

We adopt a common threat model that is widely used in Android malware research [16]. First, we assume that the adversary could trick victim users into installing the application embedded with the attack-chain code from app stores or using sideloaded APK files. The embedded APK can bypass existing state-of-the-art malware detection mechanisms, including static analysis tools and sensor status-based anomaly detection. Our evaluation results are shown in Section 6.

We also assume the user's mobile device is equipped with accelerometers and vibration motors required to construct the attack chain, both of which are commonly available on most devices. The attack chain requests only the vibration permission, which is commonly required by legitimate applications and therefore raises little user suspicion. The attack assumes that both the device vibration function and enhanced system haptic feedback are enabled.

We assume the adversary controls a separate smartphone equipped with motion sensors and a vibration motor, which serves as the receiver of the covert channel. This device is fully under the attacker's control and therefore not constrained by permission restrictions, energy limitations, or behavioral stealth requirements. To establish the covert channel, the adversary places the receiver in close physical proximity to the victim (host) device that shares the same solid surface, consistent with the assumptions adopted in prior vibration-based covert-channel research [17].

3.3 Feasibility of Call-State Haptic Signals

Our attack chain can target both system phone calls and voice communications from third-party applications. For system telephony, many commercial Android

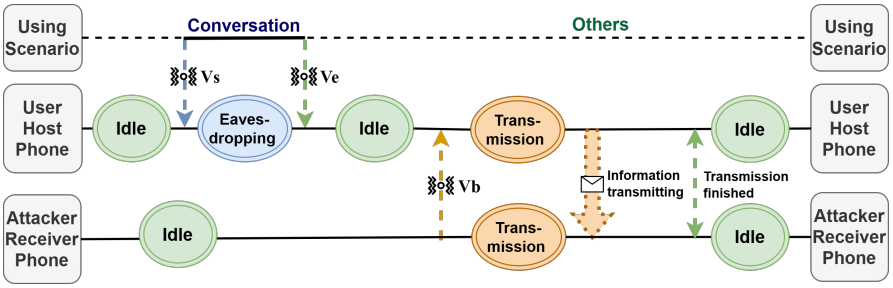

Fig. 1. Overview of the attack chain enabled by the control protocol.

smartphones provide short vibration feedback when a phone call is successfully connected. This behavior is implemented by vendor-specific dialer frameworks rather than the standard Android Open Source Project (AOSP) telephony stack. Table 2 summarizes the availability of this feature across major Android vendors. Devices from several popular manufacturers, including Samsung, OnePlus, and Xiaomi, generate deterministic haptic feedback during call connection events. Google Pixel devices support this feature through the Google Phone app, but disable it by default, while Motorola devices offer only partial or inconsistent support. These system-generated vibration events provide a reliable trigger that can be detected by motion sensors and used to activate the attack chain.

For third-party communication applications, the trigger can be implemented by binding a vibration event to the call connection button via the standard vibration API, and no additional permissions are required. This makes the behavior indistinguishable from normal UI feedback and unlikely to raise suspicion.

4 Attack Chain Design

4.1 Control Protocol Design

The attack chain on the host phone consists of three components: an ML-based eavesdropping module, a vibration-motor-driven covert channel, and a lightweight control protocol that coordinates the two. The coordination is implemented as a lightweight conditional control layer built on top of the standard accelerometer processing pipeline. The system continuously performs low-cost sensor monitoring and activates high-overhead modules only when predefined trigger patterns induced by V_s, V_e, and V_b are detected in the raw sensor data. This design ensures that resource-intensive sensing and transmission operations are executed only when necessary. As illustrated in Figure 1, both the host phone and the attacker's receiver phone remain in a low-power *idle* state for most of the time. In this state, only the lightweight sensor-processing module runs to monitor incoming accelerometer signals for the trigger patterns, while all other modules remain inactive.

During a phone conversation on the host device, the control protocol detects V_s, which corresponds to the start of the call, and activates the eavesdropping

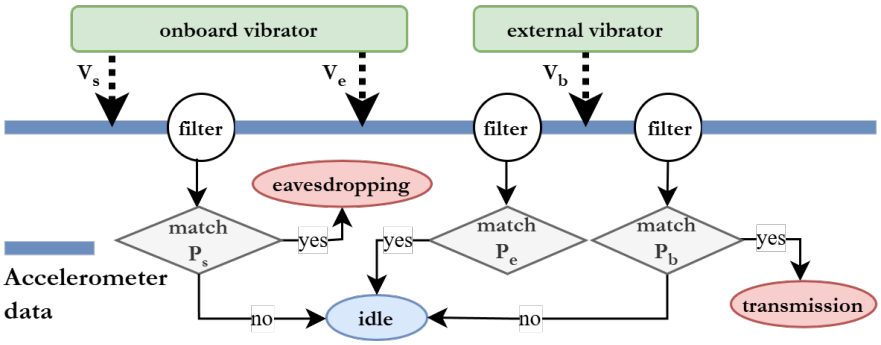

Fig. 2. Workflow of the vibration-enabled fully sensor-driven attack chain.

module to infer sensitive speech content. When the conversation ends, the protocol detects V_e, immediately terminates the module, releases allocated resources, and returns the device to the idle state. Subsequently, when the attacker places the receiver phone on a shared rigid surface within an effective distance, the receiver emits a ready signal V_b. This signal triggers the transmission module on the host phone, which transmits the inferred data through the vibration-based covert channel. After the transmission completes, both devices automatically revert to the idle state.

4.2 Energy-Efficient Filter Design

As illustrated in Figure 2, vibrations generated by the device's onboard haptic motor [8], as well as by the vibrator of a nearby smartphone placed on the same rigid surface [17], can measurably influence accelerometer readings. The application executes different control branches based on patterns detected in the raw sensor streams (a_x, a_y, and a_z). The signal-processing pipeline employs filters commonly used in sensor-based algorithms, implemented as band-pass filters with center frequency f. The filtered data are segmented using sliding time windows of width Δt, after which a threshold-based function B converts each segment into a binary representation. The resulting binary codes are continuously compared against predefined control patterns P_s, P_e, and P_b. Upon a match, the corresponding control function is executed. When no pattern is detected, the application remains in its default 'idle' operating mode, which is intentionally designed to be low-cost and non-suspicious.

Working Frequency. The working frequency f is determined by several factors: the physical working frequency range of sensors and actuators F_s and F_a, the frequency range of API limitations F_l, and the environmental and user noise range F_n. Hence, the optimal frequency f should fall within the range:

$$f \in (F_l \cap F_s \cap F_a) \setminus F_n \tag{1}$$

Time Window and Band-pass Filter. Time windows (denoted as $W(\Delta t)$) and band-pass filters are commonly used to extract desired information from

raw sensor data. The minimum width of the time window Δt must be at least $2/f$ to capture each carrier signal. Band-pass filters are widely used in signal processing. For instance, various instances of *BandPassFilter* are provided by the TarsosDSP library [29]. So the abstracted sensor data $X(f, \Delta t)$ is:

$$X(f, \Delta t) = BandPassFilter(f) \times W(\Delta t) \tag{2}$$

The next phase is to transform sensor data into binary code to detect the trigger pattern. The most common conversion process can be conducted with a threshold function 3, where the threshold is denoted as θ:

$$B(f, \Delta t) = \begin{cases} 1, \text{ if } X(f, \Delta t) \geq \theta \\ 0, \text{ if } X(f, \Delta t) < \theta \end{cases} \tag{3}$$

Threshold. Given that vibration-accelerometer covert channels have been well studied, we adopt empirically determined threshold values.

The vibration pattern is then mapped to a binary representation with proper $f, \Delta t$, and θ. For example, Android API *vibrator.vibrate(pattern)* is used to control the vibrator with a vibration pattern. This pattern consists of alternating vibration and pause intervals, with each value in the sequence representing the duration in milliseconds to either vibrate or pause. For instance, a sequence like $[100, 200, 200]$ would indicate a 100 ms vibration, followed by a 200 ms pause, then a 200 ms vibration. With $\Delta t = 100\,\text{ms}\, f = 80Hz, \theta = 0.05$, the vibration pattern $[100, 200, 200]$ would be mapped to binary code 10011.

4.3 Stealth Analysis

Static Analysis Evasion. All control logic of the attack chain is implemented entirely at the sensor-data layer and driven solely by pattern recognition over accelerometer signals. As illustrated in Pseudocode 1, the only required permission is vibration, which is non-sensitive and commonly granted to benign applications. The signal-processing techniques employed are standard in sensor-based applications, and the overall control flow closely resembles legitimate sensor-driven logic. For example, a fitness-tracking application increments a step counter upon detecting walking patterns, records stair climbing when corresponding motion features are observed, and triggers alerts via sound, light, or vibration when a fall is inferred. Our design follows the same data-driven paradigm, making the control logic difficult to distinguish from benign application behaviors.

Sensor State-based Anomaly Detection Evasion. Anomaly detectors based on sensor states, such as 6thSense [27] and SBTDDL [20], learn normal multi-sensor transitions using statistical or deep learning models. Each state is represented as a vector $X \in \mathbb{R}^n$, and the classifier C_θ is trained on labeled examples $(X^{(j)}, y^{(j)})$.

For continuously sampled sensors, such as accelerometers, state transitions are determined by comparing $\bar{a} = \frac{1}{T}\int_{t_0}^{t_0+T} a(t)\,dt$, the average reading over a time window T, against a threshold τ. As a result, the sensor-state changes

Algorithm 1 Attack Chain Implementation Code Logic

```
permission ← vibrate
mode ∈ {Default, Eavesdrop, VibrateWithPattern} ← Default
𝒲 ← Δt                                          ▷ sliding window of filtered signals
text ← ∅                                        ▷ latest extracted text
procedure OnAccelEvent(a_x, a_y, a_z)
    s ← BandPassFilter(f)(a_x, a_y, a_z)
    𝒳 ← 𝒲(Δt, s)
    binaryCode ← B(𝒳, θ)
    DefaultProcess                              ▷ always executes
    if Match(binaryCode, P_s) then
        mode ← Eavesdrop
    else if Match(binaryCode, P_e) then
        mode ← Default
    else if Match(binaryCode, P_b) then
        P ← EncodeTextToPattern(text)
        VibrateWithPattern(P)
        mode ← Default                          ▷ auto-reset after vibration
    end if
    if mode = Eavesdrop then
        text ← EavesdropProcess(a_x, a_y, a_z)  ▷ extract or update reconstructed spoken information
    end if
end procedure
```

caused by the vibration depend on the proportion of bits '1' in the trigger pattern relative to this detection threshold. Given τ, the number of '1's required to induce a state change is $\tau l/(\Delta t \cdot I_a)$, where I_a is the accelerometer reading under continuous vibration and Δt is the sampling window. Therefore, V_s, V_e, and V_b must be crafted to mimic legitimate actuator behavior so that the resulting sensor states satisfy the anomaly-evasion condition.

Existing common vibration patterns are listed in Table 3. Thanks to the wide variety of predefined vibration patterns, when converted to binary code, predefined vibration patterns exhibit a wide range of "1" proportions: from the minimal single-bit pulses of a 2 ms "tick" or 6 ms "click" up to sustained bursts of 300, 500, or even 1000 ms (i.e., sequences of l consecutive "1" bits). So when designing V_b, the translated binary code should match the proportions of "1" bits of existing legitimate vibration patterns.

5 Implementation and Evaluation

5.1 Prototype

We implement an end-to-end prototype that instantiates the control protocol described in Section 4.1. The system is deployed as two Android applications: a host application running on the victim device and a receiver application controlled by

Table 3. The common pre-defined vibration patterns

Category	Name	Patterns	Category	Name	Patterns
One-shot	short	100, 200	Long-term	heartbeat	0, 200, 500, 200
	long	200, 100		long lasting	0, 1000, 500, 1000
	TickTock	200, 100		short lasting	0, 200
	click	6, 100		SOS	Morse code*
	double click	144, 100		Rapid	0, 100, 50
	heavy click	8, 100		Symphony	0, 400, 100, 300, 100, 200
	tick	2, 100		Staccato	0, 100, 50
Continuous	short repeat	0, 200		Waltz	0, 100, 300
	long repeat	0, 1000, 500, 1000		Zig Zig	0, 100, 200
	SOS repeat	5 times repeat		Off beat	0, 300, 100, 200, 100, 400
	Siren repeat	5 times repeat		Siren	0, 500, 250
Customized	Stutter	0, 40, 60, 40, 60, 40, 120		Ripple	0, 200, 100, 150, 50, 100
	Escalating	0, 80, 100, 120, 100, 180		Telephone	0, 400, 200, 400, 1000

* Morse code for SOS is 0, 200, 100, 200, 100, 200, 300, 500, 300, 500, 300, 500, 300, 200, 100, 200, 100, 200.

the attacker (Figure 3). The host application integrates accelerometer-based inference and vibration-based transmission modules and provides a visual interface to display operating states, reconstructed content, and energy consumption.

To reduce cost and avoid unnecessary resource consumption during experiments, we follow prior work [5] and have the host phone play audio recordings of spoken numbers instead of making real phone calls.

We generate the spoken digit audio clips using Google's text-to-speech (gTTS) engine [11]. Specifically, we synthesize the English words "zero" to "nine" and convert the generated audio files into WAV format for playback in the host application. Vibration signals are generated at the beginning and end of audio playback to emulate the call-start and call-end events. The start signal V_s is implemented as [150, 150, 150], whereas the end signal V_e is implemented as a single 600 ms vibration pulse. The host application runs in idle mode by default. When the start vibration signal V_s is detected, the application switches to eavesdropping mode and activates a trained machine learning model to recover the spoken numbers from sensor observations. The recovered numbers are then encoded and stored locally. The encoding process follows the VibeComm [17] scheme, which maps each character to the last seven bits of its ASCII code with an appended check bit.

Upon receiving the transmission-trigger signal V_b, implemented as [300, 300, 300], from the receiver phone, the host application switches to transmission mode and emits vibration patterns according to the stored binary code at the rate of one bit per 300 ms ($\approx$ 3.3 bits/s). On the receiver side, the application records the incoming vibration signals, decodes the binary code with threshold $\theta = 0.05 m/s^2$, and displays the decoded numbers. After the transmission is completed, both applications return to idle mode.

Machine Learning Models. We trained three representative models with increasing computational complexity and capability. The first is a lightweight in-house 1D CNN designed specifically for ten-digit recognition, with a TFLite model size of 45.0 kB. The second is a medium-scale 1D ResNet-50–based model,

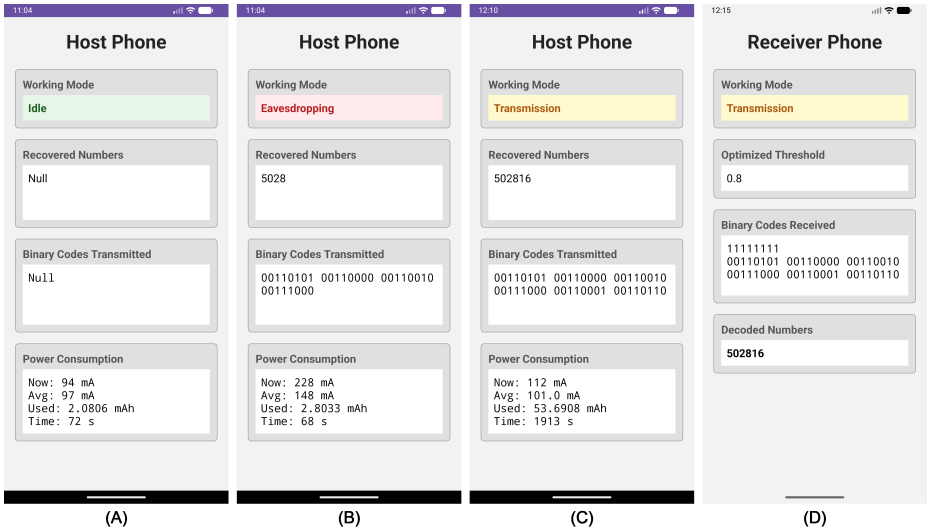

Fig. 3. Screenshots of the proposed prototypes: (A) initial/idle mode of the host phone, (B) eavesdropping mode of the host phone, (C) transmission mode of the host phone, and (D) transmission mode of the receiver phone.

representing a stronger yet still practical architecture with approximately 23M parameters and a TFLite model size of 63.9 MB. The third is a large 1D ResNet-152–based model with deeper blocks and wider channels, resulting in approximately 120–140M parameters and a TFLite model size of 612.8 MB, which is the most computationally intensive model evaluated in this study. For data collection, we recorded 100 samples for each digit. Each sample is collected for approximately 0.8 s at a sampling rate of 200 Hz. The collected data is randomly partitioned into a training set (70%) and a test set (30%). For each model, we conducted 11 independent train/test splits and report the mean test accuracy across three models. All three trained models achieve 99.5% accuracy on the test dataset. Specifically, the averaged accuracies are 99.4% for the CNN, 99.4% for ResNet-50, and 99.7% for ResNet-152.

5.2 Energy Consumption

Experimental Setup. Battery current draw is logged via Android's BatteryManager interface [4], which reports instantaneous battery current as provided by the device's fuel gauge. All measurements are conducted on the same physical device (Xiaomi Mi 10 Ultra) under identical system configurations. We follow the common setup for mobile energy consumption analysis in the literature [7] to measure an app's energy consumption using minimal background application activity as the baseline. The screen brightness is set to the lowest level, and the system volume is set to the maximum level so that the played speech signals produce sufficiently strong vibrations for reliable inference.

Evaluation Metrics. Existing energy-based anomaly detectors are not publicly available. However, these approaches consistently rely on aggregate energy

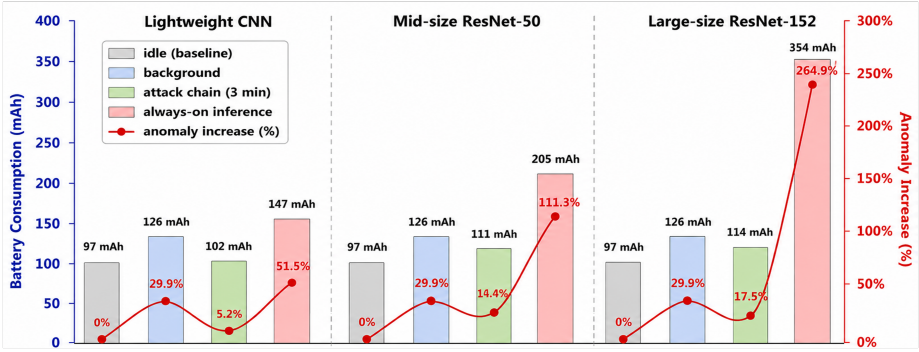

Fig. 4. Comparison of energy consumption and anomaly increase under different execution modes across lightweight CNN, ResNet-50, and ResNet-152 models.

consumption over a time window as a key time-domain feature for distinguishing abnormal behaviors. To avoid implementation-specific bias, we therefore use the increase in aggregate energy consumption relative to the baseline as a detector-independent evaluation metric.

Recent surveys indicate that about 52% of Americans spend fewer than 15 minutes per day on voice calls [35]. Meanwhile, overall smartphone usage averages approximately 4.5 hours per day [3]. To approximate realistic usage conditions, we measure energy consumption using cumulative battery usage over a fixed 1-hour window with the screen kept on. Within this window, a 3-minute audio segment is played to emulate a typical phone call. The 1-hour duration is sufficiently long to capture steady-state energy behavior while avoiding transient fluctuations caused by short bursts of computation.

Results. As shown in Figure 4, the idle baseline energy consumption is 97 mAh. Under realistic screen-on conditions with common background activities enabled, such as messaging synchronization, web browsing, email updates, and system services, the aggregate energy consumption increases to approximately 126 mAh. Our attack chain reduces abnormal energy consumption to a range comparable to common background activity, i.e., from 102 mAh to 114 mAh.

In contrast, continuously running machine-learning models introduces noticeable energy-consumption anomalies, even when attackers sacrifice attack capability by adopting lightweight models. In our experiments, the continuously running lightweight model increases energy consumption by 51.5%. This gap becomes more pronounced when more powerful models are used to achieve stronger attack performance. For the ResNet-50-based model, continuous execution leads to a 111.3% increase in energy consumption, whereas our attack chain limits the increase to 14.4%. Similarly, for the ResNet-152-based model, our attack chain reduces the energy overhead from 264.9% to 17.5%.

5.3 Attack Success Rate

Experimental Setup. We configure a Samsung S9 as the receiver device. The overall success rate of the attack chain is primarily determined by the reliability

of the side channel and the covert channel. The side channel remains stable because it operates entirely on the same device. In contrast, the performance of the covert channel depends on the properties of the transmission medium, particularly the surface material and the distance between devices. Therefore, we evaluate the attack chain across seven commonly encountered surfaces and 4 separation distances as listed in Table 4. All phones were placed flat on the surface with the screen facing upward. Distances were measured as the straight-line distance between the centers of the phones.

Evaluation Metrics. We adopt attack success rate (ASR) as the primary performance metric, distinguishing between two complementary measures: the side-channel ASR ($\mathrm{ASR_{SC}}$) and the attack chain ASR ($\mathrm{ASR_{AC}}$). These two metrics capture different aspects of the attack pipeline. $\mathrm{ASR_{SC}}$ measures how accurately the side-channel module recovers the victim's input, whereas $\mathrm{ASR_{AC}}$ evaluates end-to-end reliability. Specifically, whether the covert channel is successfully triggered and whether the inferred content is correctly transmitted to the attacker. Importantly, $\mathrm{ASR_{SC}}$ does not directly bind $\mathrm{ASR_{AC}}$: the covert channel relays whatever output the side-channel module produces, irrespective of its correctness. Formally, $\mathrm{ASR_{SC}}$ is defined as:

$$\mathrm{ASR_{SC}} = \frac{1}{M} \sum_{m=1}^{M} l_m \left(\frac{d_m^{\mathrm{correct}}}{d_m^{\mathrm{total}}} \right) \tag{4}$$

where M is the total number of trials, $l_m \in \{0, 1\}$ indicates whether the side channel was successfully triggered in trial m, d_m^{correct} is the number of digits correctly inferred in that trial, and d_m^{total} is the total number of target digits.

$\mathrm{ASR_{AC}}$ is defined as:

$$\mathrm{ASR_{AC}} = \frac{1}{N} \sum_{j=1}^{N} i_j \left(\frac{d_j^{\mathrm{correct}}}{d_j^{\mathrm{total}}} \right) \tag{5}$$

where N is the number of trials, $i_j \in \{0, 1\}$ indicates whether the covert channel was successfully triggered in trial j, d_j^{correct} is the number of digits correctly transmitted in that trial, and d_j^{total} is the total number of digits transmitted.

Results. Table 4 summarizes the experimental results. $\mathrm{ASR_{SC}}$ achieves consistently high accuracy across all conditions. In particular, the trigger success rate reaches 100%, while the remaining errors originate entirely from the side-channel inference stage, resulting in an overall success rate of 98%. These results confirm that the vibration-based event-driven mechanism is highly reliable and that motion sensor-based eavesdropping remains viable under sampling-rate constraints when supported by sufficiently expressive machine learning models.

For $\mathrm{ASR_{AC}}$, when two smartphones are placed in direct contact ($0cm$), vibrations couple directly through the phone bodies and the attack achieves over 90% success across all tested surfaces. When devices are separated, and vibrations must propagate via a shared surface, performance becomes highly sensitive to surface material. On rigid surfaces such as metal, wood, and marble, the success

Table 4. ASR_{SC} and ASR_{AC} across different distances and surface types.

Surface	Type	$\mathbf{ASR_{SC}}$	$\mathbf{ASR_{AC}}(0cm)$	$\mathbf{ASR_{AC}}(5cm)$	$\mathbf{ASR_{AC}}(15cm)$	$\mathbf{ASR_{AC}}(30cm)$
Wood desk	Rigid	98%	100%	100%	100%	100%
Marble countertop	Rigid	98%	100%	100%	100%	100%
Metal surface	Rigid	98%	100%	100%	100%	100%
Plastic table	Semi-rigid	98%	100%	100%	100%	96.7%
Wood + towel	Damped	98%	90%	0%	0%	0%
Mattress/Sofa	Soft	98%	90%	0%	0%	0%
Handheld	Soft	99%	93.3%	N/A	N/A	N/A

rate remains 100% even at 30 cm. However, soft materials severely attenuate vibration signals, dropping the success rate to 0% on a sofa or towel at just 5 cm. While placing two devices in direct contact may raise suspicion in practice, the attack can be conducted covertly from a realistic distance on a rigid surface.

6 Stealthiness Evaluation

6.1 Static Analysis Evasion

Experiment Setup. To evaluate the stealth advantages of our call-status inference design, we implement two Android prototypes corresponding to the trigger mechanisms used in the attack chain. Both prototypes share identical payload logic and attack behavior, differing only in how call events are detected.

The telephony-based prototype directly monitors call-state transitions using `TelephonyManager`, which requires the `READ_PHONE_STATE` permission. In contrast, the vibration-based prototype relies on the haptic feedback generated during call events. It declares only the `VIBRATE` permission and infers call-state transitions by monitoring motion signals through `SensorManager` using the `TYPE_ACCELEROMETER` sensor.

Evaluation Tools. We analyze the static-analysis visibility of both prototypes using MobSF (Mobile Security Framework) [23], an open-source automated Android security analysis platform widely used for Android application security.

Results. Both trigger mechanisms are observable at the API level under MobSF, as illustrated in Figure 5. However, their security implications differ significantly. The telephony-based design introduces the `READ_PHONE_STATE` permission, which MobSF classifies as a *Dangerous* permission because it allows access to sensitive telephony information. Applications requesting this permission are therefore more likely to attract scrutiny during automated analysis.

In contrast, the vibration-based design requires only the `VIBRATE` permission, a *Normal* Android permission. Prior work found that vibration is used by 71% of the top 30 apps across categories [34], making it a familiar behavior for users.

6.2 Anomaly Detection

Evaluation Tools. Prior work, such as 6thSense [27] and SBTDDL [20], has demonstrated effective detection of sensor-based anomalous behaviors. However, neither system is publicly available. Therefore, we reproduced their detection

☰ APPLICATION PERMISSIONS

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.READ_PHONE_STATE	dangerous	read phone state and identity	Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and so on.
[redacted].telephonytrigger.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION	unknown	Unknown permission	Unknown permission from android reference

☰ APPLICATION PERMISSIONS

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.VIBRATE	normal	control vibrator	Allows the application to control the vibrator.
[redacted].vibrationacceltrigger.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION	unknown	Unknown permission	Unknown permission from android reference

Fig. 5. MobSF analysis results. Upper: report showing the app invoking the READ PHONE STATE API. Lower: report showing the app invoking the vibration API.

pipelines as described in the original papers and trained the detectors from scratch on newly collected datasets.

Data Collection. We followed the data-collection protocol described in SBTDDL [20]. Each activity session lasted five minutes, with continuous sensors sampled at 1 Hz and state-change sensors sampled at 5 Hz. In total, we collected 31 activity sessions. To build a realistic dataset, we recruited five volunteers and collected 9,300 minutes of sensor data from four smartphones during normal daily activities. Following the claim in SBTDDL that the detector is device-independent, we trained the detector as a global model using data aggregated across all four devices. The use of multiple devices was intended solely to accelerate the data collection process. Furthermore, although neither 6thSense nor SBTDDL explicitly considered vibration artifacts in their datasets, we incorporated vibration signals into the collected data to ensure that the detectors were exposed to realistic haptic events.

Results. We executed the attack chain for 100 hours under realistic daily-usage scenarios. During this period, the synchronization and termination signals, V_b and V_e, were consistently classified as benign by the trained detectors. These signals correspond to common system-predefined vibration patterns, and labeling them as anomalous would lead to an unacceptably high false-positive rate in real-world usage. We further evaluated 10 different V_s signal patterns that follow the design discipline described in Section 4.3. None of these patterns was flagged as anomalous by the sensor-anomaly detector, indicating that the covert vibration signaling remains indistinguishable from benign system-generated vibration.

7 Discussion

7.1 Fuzzing Test

Automated fuzzing frameworks used in malware analysis attempt to expose suspicious behaviors by injecting stimuli in emulated environments and monitor-

ing abnormal API calls or resource-usage patterns. In our design, attack-chain state transitions are controlled by three vibration-based signals, suggesting that fuzzing these signals could potentially reveal suspicious behavior.

However, in emulated environments, all behaviors triggered by these signals appear benign. The signal V_e simply transitions the application from the eavesdropping state to idle, producing no observable side effects. The signal V_s activates a local machine-learning inference module operating solely on sensor data, which is indistinguishable from legitimate sensor-driven ML execution. Because emulators do not generate sound-induced accelerometer signals, the inference model produces no output. Similarly, V_b produces no effect because no physical vibration motor exists; the emulator executes the corresponding code paths without generating physical signals or errors.

Even on real devices, effective fuzzing remains challenging. The eavesdropping module requires active audio playback to generate meaningful sensor responses, valid inference outputs, and transmissible vibration signals. Without these conditions, V_s produces no valid output, and V_b generates no vibration. Ensuring such conditions significantly complicates the fuzzing setup and increases the difficulty of constructing effective test scenarios.

7.2 Practical Constraints

Our design infers call-state transitions from haptic feedback generated by the operating system or voice-call applications. However, Android allows users to disable vibration feedback, and some may do so for comfort or battery savings. In such cases, the proposed inference mechanism would not operate. Nevertheless, given the large global smartphone user base, even a subset of users with haptic feedback enabled represents a substantial attack surface.

Limitations. Our evaluation trains the ML models using accelerometer data collected from a single device and does not investigate cross-device generalizability or transferability. Since different smartphones may exhibit distinct vibration characteristics, the eavesdropping models may require retraining when deployed on other devices. However, improving the robustness or generalizability of the side-channel inference model is beyond the primary scope of this work. Prior studies have shown that cross-device robustness can be improved through sufficiently large and diverse training datasets [16].

In addition, the performance of the covert channel may be affected by environmental factors such as surface materials and device cases, both of which can influence vibration propagation. In contrast, device cases have a relatively limited impact on the eavesdropping module because sound-induced vibrations primarily propagate through the smartphone body itself.

7.3 Mitigation

A simple and effective mitigation is to disable system haptic feedback during call initiation and termination across both system components and third-party applications. Since our control protocol relies on the vibration patterns generated by

these haptic events as trigger signals, removing them would eliminate the underlying physical control channel and prevent reliable trigger detection. However, such a mitigation weakens an important interaction mechanism widely used in mobile systems to provide immediate tactile confirmation for user actions. As a result, disabling haptic feedback may negatively impact usability, accessibility, and overall user experience, particularly in silent or attention-limited scenarios where tactile cues complement visual or auditory feedback.

8 Conclusion

This paper shows that the practicality of motion-sensor attacks can be restored through a lightweight control protocol implemented entirely at the sensor-data layer. By exploiting vibration-induced accelerometer artifacts, the proposed design coordinates call-context inference, covert-channel synchronization, and data exfiltration without sensitive APIs or explicit communication channels. The resulting attack chain maintains high reliability while significantly reducing abnormal energy consumption, demonstrating that fully sensor-driven attacks remain feasible under modern Android defenses. Our findings highlight that the control plane of sensor-based attacks is an overlooked security surface. Effective defenses must therefore consider actuator–sensor interactions rather than relying solely on permission checks or isolated sensor monitoring.

Acknowledgment

This work was supported in part by NSF IIS-2014552, DGE-1565570, and the Ripple University Blockchain Research Initiative. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

1. Aboleaze, M.A., Elnaggar, A.: Reducing memory references for fft calculation. In: Proc. of the International Conference on Computer Design. pp. 26–28 (2006)
2. Al-Haiqi, A., Ismail, M., Nordin, R.: A new sensors-based covert channel on android. The Scientific World Journal **2014**(1), 969628 (2014)
3. Alexus Bazen: Cell phone statistics 2025-consumeraffairs (2025), `https://www.consumeraffairs.com/cell_phones/cell-phone-statistics.html`, accessed: 2026-03-10
4. Android Developers: Batterymanager. `https://developer.android.com/reference/android/os/BatteryManager`, accessed: 2025-03
5. Ba, Z., Zheng, T., Zhang, X., Qin, Z., Li, B., Liu, X., Ren, K.: Learning-based practical smartphone eavesdropping with built-in accelerometer. In: NDSS. vol. 2020, pp. 1–18 (2020)
6. Cai, L., Chen, H.: Touchlogger: inferring keystrokes on touch screen from smartphone motion. In: 6th USENIX Workshop on Hot Topics in Security (HotSec 11) (2011)

7. Carroll, A., Heiser, G.: An analysis of power consumption in a smartphone. In: 2010 USENIX Annual Technical Conference (USENIX ATC 10) (2010)
8. Casolare, R., Martinelli, F., Mercaldo, F., Santone, A., et al.: Colluding covert channel for malicious information exfiltration in android environment. In: ICISSP. pp. 811–818 (2021)
9. Cathis, A., Li, G., Wei, S., Orshansky, M., Tiwari, M., Gerstlauer, A.: Sok paper: Power side-channel malware detection. In: Proceedings of the International Workshop on Hardware and Architectural Support for Security and Privacy 2024. pp. 1–9 (2024)
10. Dadafshar, M.: Accelerometer and gyroscopes sensors: operation, sensing, and applications. Maxim Integrated [online] (2014)
11. Durette, P.N.: gtts: Google text-to-speech. `https://github.com/pndurette/gTTS` (2023)
12. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS) **32**(2), 1–29 (2014)
13. Gao, M., Liu, Y., Chen, Y., Li, Y., Ba, Z., Xu, X., Han, J.: Inertiear: Automatic and device-independent imu-based eavesdropping on smartphones. In: IEEE INFOCOM 2022-IEEE Conference on Computer Communications. pp. 1129–1138. IEEE (2022)
14. Gasior, W., Yang, L.: Exploring covert channel in android platform. In: 2012 international conference on cyber security. pp. 173–177. IEEE (2012)
15. Google: Sensors overview: Sensor rate limiting. `https://developer.android.com/guide/topics/sensors/sensors_overview` (Accessed, 8-8-2024)
16. Hu, P., Zhuang, H., Santhalingam, P.S., Spolaor, R., Pathak, P., Zhang, G., Cheng, X.: Accear: Accelerometer acoustic eavesdropping with unconstrained vocabulary. In: 2022 IEEE S&P. pp. 1757–1773. IEEE (2022)
17. Hwang, I., Cho, J., Oh, S.: Vibecomm: Radio-free wireless communication for smart devices using vibration. Sensors **14**(11), 21151–21173 (2014)
18. Kim, H., Smith, J., Shin, K.G.: Detecting energy-greedy anomalies and mobile malware variants. In: Proceedings of the 6th international conference on Mobile systems, applications, and services. pp. 239–252 (2008)
19. Liccardi, I., Pato, J., Weitzner, D.J.: Improving mobile app selection through transparency and better permission analysis. Journal of Privacy and Confidentiality **5**(2), 1–55 (2014)
20. Manimaran, S., Sastry, V., Gopalan, N.: Sbtddl: A novel framework for sensor-based threats detection on android smartphones using deep learning. Computers & Security **118**, 102729 (2022)
21. Manimaran, S., Sastry, V., Gopalan, N.: Stmad: sensor-based threat's mitigation on smartphones using allowlist and denylist. The Journal of Supercomputing **78**(14), 16336–16363 (2022)
22. Mayrhofer, R., Stoep, J.V., Brubaker, C., Kralevich, N.: The android platform security model. ACM Transactions on Privacy and Security (TOPS) **24**(3), 1–35 (2021)
23. MobSF Project: Mobile security framework (mobsf). `https://github.com/MobSF/Mobile-Security-Framework-MobSF` (2025), accessed: 2026-02-26
24. Muhammad, Z., Anwar, Z., Javed, A.R., Saleem, B., Abbas, S., Gadekallu, T.R.: Smartphone security and privacy: a survey on apts, sensor-based attacks, side-channel attacks, google play attacks, and defenses. Technologies **11**(3), 76 (2023)

25. Roy, N., Gowda, M., Choudhury, R.R.: Ripple: Communicating through physical vibration. In: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). pp. 265–278 (2015)
26. Shekhar, S., Dietz, M., Wallach, D.S.: {AdSplit}: Separating smartphone advertising from applications. In: 21st USENIX Security Symposium (USENIX Security 12). pp. 553–567 (2012)
27. Sikder, A.K., Aksu, H., Uluagac, A.S.: {6thSense}: A context-aware sensor-based attack detector for smart devices. In: 26th USENIX Security Symposium. pp. 397–414 (2017)
28. Sikder, A.K., Petracca, G., Aksu, H., Jaeger, T., Uluagac, A.S.: A survey on sensor-based threats and attacks to smart devices and applications. IEEE Communications Surveys & Tutorials **23**(2), 1125–1159 (2021)
29. Six, J., Cornelis, O., Leman, M.: TarsosDSP, a Real-Time Audio Processing Framework in Java. In: Proceedings of the 53rd AES Conference (2014)
30. Song, Y., Kukreti, M., Rawat, R., Hengartner, U.: Two novel defenses against motion-based keystroke inference attacks. arXiv preprint arXiv:1410.7746 (2014)
31. StatCounter Global Stats: Mobile & tablet android version market share worldwide — statcounter global stats. `https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide/#monthly-202412-202601` (2026), accessed: 2026-02-02
32. Su, W., Liu, D., Zhang, T., Jiang, H.: Towards device independent eavesdropping on telephone conversations with built-in accelerometer. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies **5**(4), 1–29 (2021)
33. Wang, P.: Ne10 FFT Feature: Radix-3 and Radix-5 FFT are Supported, NEON Optimization Significant Performance Improvement by NEON-Optimization. ARM Community Blog (Operating Systems Blog) (Jan 2015), `https://developer.arm.com/community/arm-community-blogs/b/operating-systems-blog/posts/ne10-fft-feature-radix-3-and-radix-5-fft-are-supported-neon-optimization-significant-performance-improvement-by-neon-optimization`, accessed: 2026-03-12
34. Wang, Y., Luo, B., Li, F.: Beyond conventional triggers: Auto-contextualized covert triggers for android logic bombs. In: Network and Distributed System Security (NDSS) Symposium (2026)
35. YouGov: How long do americans talk on calls in a day? (Jul 2024), `https://yougov.com/en-us/articles/49894-how-long-do-americans-talk-on-calls-in-day`, survey of U.S. adults on daily voice call duration
36. Zhang, L., Pathak, P.H., Wu, M., Zhao, Y., Mohapatra, P.: Accelword: Energy efficient hotword detection through accelerometer. In: Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services. pp. 301–315 (2015)
37. Zhang, S., Liu, Y., Gowda, M.: I spy you: Eavesdropping continuous speech on smartphones via motion sensors. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies **6**(4), 1–31 (2023)