EarFisher: Detecting Wireless Eavesdroppers by Stimulating and Sensing Memory EMR

Cheng Shen[1], Jun Huang[2]*
[1]*Peking University,* [2]*Massachusetts Institute of Technology*

Abstract

Eavesdropping is a fundamental threat to the security and privacy of wireless networks. This paper presents EarFisher – the first system that can detect wireless eavesdroppers and differentiate them from legitimate receivers. EarFisher achieves this by stimulating wireless eavesdroppers using bait network traffic, and then capturing eavesdroppers' responses by sensing and analyzing their memory EMRs. Extensive experiments show that EarFisher accurately detects wireless eavesdroppers even under poor signal conditions, and is resilient to the interference of system memory workloads, high volumes of normal network traffic, and the memory EMRs emitted by coexisting devices. We then further propose a method to detect eavesdropper's countermeasure, which deliberately emits strong memory EMR to interfere with EarFisher's detection.

1 Introduction

Rendered by the broadcast characteristic of wireless medium, eavesdropping has been a fundamental threat to the security and privacy of wireless networks ever since the invention of wireless communication. While significant cryptographic research has been devoted to tackling this threat, in this paper, we take an orthogonal angle to explore the feasibility of wireless eavesdropper detection. A security primitive capable of this task is essential in a wide range of scenarios. First, in wireless networks serving public areas (e.g., airports, campus, malls, etc), Layer-2 encryption is commonly disabled to facilitate open access. Second, encryption algorithms are subject to extensive side channel analysis [7, 8, 11, 15, 16, 21, 22], which allow attackers to decipher encryption keys. Third, cryptographic protocols themselves often suffer fatal flaws that are difficult to identify before universal adoption. For example, in 2017, researchers uncovered that the four-way handshake of WPA2 is vulnerable to the key re-installation attack [35], which allows eavesdroppers to compromise encryption keychains. The flaw had been present since the release of 802.11i in 2004, leaving billions of Wi-Fi users potentially exposed to eavesdropping for more than 13 years.

Beyond complementing encryption schemes, an eavesdropper detector can be an essential building block of a secure network. For example, in quantum networks, legitimate receivers can detect eavesdroppers by leveraging the quantum physic law, where the state of a quantum bit 'collapses' whenever it is intercepted. Quantum key distribution protocols [9] use this law to verify the confidentiality of encryption keys, which leads to fundamentally assured communication security in quantum networks.

Unfortunately, to date, there has been no effective method to detect eavesdroppers in wireless networks. Unlike quantum eavesdroppers, wireless eavesdroppers can be completely passive without actively transmitting or altering signals in the air. Recent studies exploit the RF leakages of radio circuits to detect wireless receivers [13, 24, 26, 27, 32, 34, 36]. However, such leakages are extremely weak, limiting detection range to only a few feet. More importantly, because all wireless radios emit leakages during signal reception, this method cannot differentiate eavesdroppers from legitimate receivers.

In this paper, we present EarFisher – the first system that can detect wireless eavesdroppers and differentiate them from legitimate receivers. The key idea is based on the observation that, unlike legitimate receivers who drop others' packets in network interface cards (NICs), only eavesdroppers digest all packets in their CPU-memory systems. Inspired by this observation, EarFisher stimulates eavesdroppers by transmitting a flow of *bait* packets forged with a virtual receiver address, and then detects eavesdroppers by sensing the surge of their electromagnetic radiations (EMRs) when they write baits into their memories. Recent studies show that the multi-channel architecture of modern memories amplifies memory EMR [17], which helps EarFisher extend detection range.

To realize this idea, we tackle four key challenges. First, when multiple devices having the same memory frequency coexist in an environment, their memory EMRs mix in frequency spectrum, making it difficult to accurately sense them separately. Second, memory workloads of operating systems

*Corresponding author: junhuang@mit.edu.

and applications also produce memory EMRs, which are difficult to distinguish from the eavesdropper's responses when the memory activities coincidentally occur at the same time of bait packet transmissions. Third, despite the amplification of multi-channel architecture, memory EMR is still very weak, requiring a long time of stimulus to trigger a sufficiently strong response at the eavesdropper. However, intensive bait packet transmissions in a large time window can block the wireless channel, and may present a distinguishable traffic pattern that can be noticed by crafty eavesdroppers. Fourth, eavesdroppers knowing the design and presence of EarFisher may deliberately write its memory, which will produce strong memory EMR that interferes with EarFisher's detection.

To address these challenges, EarFisher employs new signal processing algorithms to sense and separate the memory EMRs of different devices, obscures and disguises stimulus traffic as short bursts of normal packets, incorporates statistical tool to tolerate the interfering EMRs produced by system memory workloads, and exploits a fundamental dilemma of eavesdroppers to detect deliberate writing-based countermeasure. Extensive experiments show that EarFisher accurately detects eavesdroppers even under poor signal conditions, and is resilient to the interference of system memory workloads, high volumes of normal network traffic, and the memory EMRs of coexisting devices. We then further demonstrate EarFisher's effectiveness in a real testbed, where three EarFisher nodes are deployed to monitor an indoor area of 1600 ft^2. Experiment results show that EarFisher reliably detects eavesdroppers placed at different locations despite the complexity of indoor environments, such as the block of walls.

2 Related Work

Cryptographic attacks. Since the invention of wireless communication, encryption has been the primary security measure to combat eavesdropping. However, extensive research shows that eavesdroppers can compromise encryption based on a variety of side-channel attacks [7, 8, 15, 16, 21, 22]. In particular, Camurati et al. [11] show that the EM leakage of the processor on wireless chips can be inadvertently amplified by RF front-end, allowing an eavesdropper to decipher the encryption key from a distance.

Moreover, cryptographic protocols themselves often suffer fatal flaws that are difficult to identify before universal adoption. For example, Wired Equivalent Privacy (WEP), a security protocol ratified as a part of 802.11 in 1997, was found to have fatal flaws in 2001 [14]. WEP was then superseded by Wi-Fi Protected Access (WPA) in 2004, but history repeated itself. In 2017, researchers demonstrated that the four-way handshake of WPA has a fatal vulnerability, which allows eavesdroppers to compromise the encryption keychain using key reinstallation attacks [35]. From 2004 to 2017, the vulnerability of WPA left billions of Wi-Fi users potentially exposed to eavesdropping for more than 13 years.

Eavesdropper detection. Prior eavesdropper detectors commonly rely on sensing RF signals leaked from the front-end circuit of wireless receiver. This method was first proposed in [27, 36] to detect primary receivers in cognitive radio networks, and then extended to UWB, WiMAX, and MIMO channels to detect hidden radios [24, 26]. Ghostbuster [13] further augments this method using spatial cancellation of interference and frequency cancellation of signal artifacts, which allow it to extract leakages under poor signal conditions in the presence of ongoing wireless transmissions. However, Ghostbuster still suffers limited detection range (less than 1m for COTS Wi-Fi receivers) because of the weak power of leakage from RF circuit.

Recent studies [32, 34] show that superheterodyne and superregenerative receivers can be detected from a longer distance by exploiting the known characteristics of their RF leakages. However, these techniques are highly dependent on the receiver architecture. Unfortunately, superheterodyne and superregenerative receivers are commonly used in low-power wireless remote, but are far less common in mainstream wireless communication systems such as Wi-Fi.

More importantly, because all radios emit RF leakages during signal reception, existing eavesdropper detectors suffer a common limitation as they cannot differentiate eavesdroppers from legitimate receivers. To sidestep this limitation, Ghostbuster assumes a threat model where the number of legitimate receivers is known a priori [13]. This assumption restricts its usability to a very narrow range of specific scenarios.

EM side-channels. Recently, researchers leverage the EM side channels of CPU and memory for attestation [29], memory profiling [30], and malware detection [18, 25, 37]. Different from these passive analysis, EarFisher features a new paradigm that actively stimulates memory EMR. Moreover, EarFisher employs a signal processing algorithm that bases on fine-grained measurement and theoretical characterization of memory clock, which makes it possible to not only extract weak memory EMRs under poor signal conditions, but also separate and track individual memory EMRs when multiple devices having the same memory frequency coexist in a crowded environment.

3 Threat Model

A wireless eavesdropper is a malicious receiver who sniffs on other devices' packets. Typically, eavesdropping can be implemented by modifying the device driver to enable *monitor mode*, in which a wireless chip transfers all received packets to the host. A recent study shows that most drivers on major operating systems (e.g., Linus, Windows, macOS) support monitor mode by default [1].

By eavesdropping on network traffic, the attacker's goal is to gather sensitive data, such as personal and business related information, or secrets necessary to enable decipher and

man-in-the-middle attacks. Typically, wireless chips are integrated with a microcontroller and a small RAM of at most a couple of MiBs. The on-chip system is tasked for simple computation such as checking receiver address and verifying packet checksum, but is far from capable of security- and privacy-intrusive processing. As a result, the eavesdropper must digest sniffed packets in the CPU-memory system of the host.

To this end, the wireless chip of the eavesdropper needs to write all sniffed packets to the memory of the host. This is typically under the control of DMA, and will produce EMR when sniffed packets are transferred over memory bus. Specifically, for SDRAMs, memory EMR can be observed as a radio signal around the frequency of memory clock. Because DDR uses double pumping, the clock frequency is a half of memory speed. In practice, the clock frequency of a modern DDR has 13 possible values ranging from 200 MHz of DDR2-400 to 1600 MHz of DDR4-3200. To sense memory EMR, one can scan DDR frequencies one by one, or use a group of sensors to monitor multiple frequencies in parallel. In the rest of this paper, we assume the memory clock frequency of the eavesdropper is known.

4 Characterizing Memory EMR

In this section, we present measurements and model to characterize memory EMR. Our measurements are conducted on two laptops of DDR3-1600 and DDR4-2133, respectively. A BladeRF with an omni-directional 5 dBi antenna is employed as the receiver.

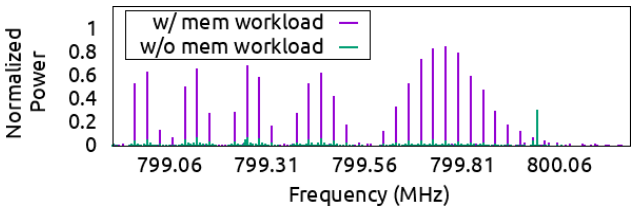
4.1 Spectrum Pattern

Measurement-based characterization. To measure the frequency spectrum of a device's memory EMR, we take FFT over an 1s window of signals captured around the clock frequency of the device's memory. To study how memory workload impacts on memory EMR, we created a process to write memory intensively[1], and then compare the spectrum patterns measured before and after the start of the process.

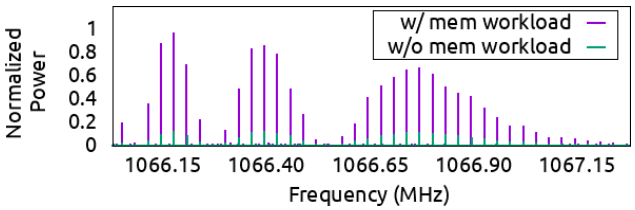
As shown in Fig. 1, we observe that the frequency spectrum of memory EMR features a series of energy peaks distributed over a side-band of about 1 MHz below the frequency of memory clock. In the presence of intensive memory workload, the amplitudes of all energy peaks increase simultaneously and significantly. To further characterize the spectrum pattern, we examine the auto-correlation of peak frequencies and find that the frequency interval between consecutive peaks is constant. Specifically, for the DDR3-1600 and the DDR4-2133, the frequency intervals are 31.8 KHz and 31.2 KHz, respectively.

Theoretical characterization. We then further characterize memory EMR based on the theoretical model of memory

[1]To write memory directly, we used SSE instructions to bypass cache.



(a) DDR3-1600.



(b) DDR4-2133.

Figure 1: The spectrum pattern of memory EMR with and without memory workload.

clock. Because memory clock injects fluctuating current into memory bus, it produces EMR at the clock frequency. As memory reads/writes are performed at clock edges, they amplify EMR amplitude of memory clock but will not affect the pattern of frequency spectrum.

The simplest form of a clock is a sine wave of which the energy is all concentrated at clock frequency. However, this leads to a high EMR intensity that may violate regulatory requirement for electromagnetic compatibility. To walk-around this issue, modern clock generators use spread spectrum techniques to reshape the energy distribution of clock. Denote an unspread clock as,

$$V_{clk}(t) = A\cos(2\pi f_0 t),$$

where f_0 is the frequency of memory clock. A spread spectrum clock is the frequency modulation of $V_{clk}(t)$, which can be expressed as,

$$V_{ssc}(t) = A\cos(2\pi f_0 t + \frac{\Delta f}{f_m}\sin(2\pi f_m t)), \tag{1}$$

where f_m and Δf are the modulation frequency and peak frequency deviation, respectively. By expanding Eqn. 1 using the Jacobi-Anger expansion, the frequency spectrum of $V_{ssc}(t)$ can be expressed as [10],

$$\|\mathcal{F}_{ssc}(f)\| \stackrel{\text{def}}{=} \|\sum_n J_n(\frac{\Delta f}{f_m})(\delta(f - f_0 + nf_m) - \delta(f - f_0 - nf_m))\|.$$

where $J_n(\cdot)$ is the Bessel function of the first kind, and $\delta(\cdot)$ is the Dirac delta function.

To maintain a stable synchronization between memory and memory controller, $V_{ssc}(t)$ needs to be band-pass filtered before being used as a memory clock. Typically, the band-pass filter removes frequency components higher than memory clock frequency and then imposes a low cutoff to limit the

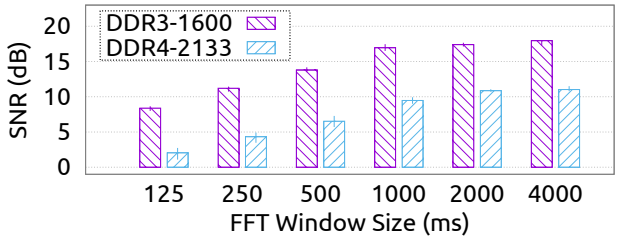


Figure 2: The SNR of received memory EMR as a function of FFT window size.

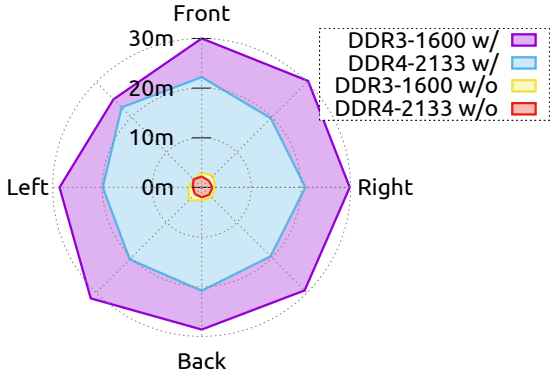


Figure 3: The 3 dB range of memory EMRs with and without memory activities.

maximum frequency deviation. After band-pass filtering, the frequency spectrum of $V_{ssc}(t)$ is transformed to,

$$\|\mathcal{F}_{ssc}(f)\| \stackrel{\text{def}}{=} \|\mathcal{A}(f)\sum_{n} J_{\text{n}}(\frac{\Delta f}{f_m})\delta(f-f_0+nf_m)\|, \quad (2)$$

where $\mathcal{A}(f)$ is the impulse response of the band-pass filter.

Summary of characteristics. Eqn. 2 indicates that the energy of memory EMR is non-zero only at $f_0 - nf_m$, which interpreted the spectrum pattern shown in Fig. 1. In other words, memory EMR is composed of a series of *sub-clock components*, where the frequency interval between consecutive sub-clocks is f_m, and the first sub-clock is at f_0.

4.2 Directivity and Range

We then place receiver at different directions to measure the *3 dB range* of memory EMR, which is defined as the maximum distance from which the SNR of received memory EMR is higher than 3 dB. In this experiment, we calculate SNR of memory EMR using the sub-clock of the highest power.

It is worth noting that the SNR of received memory EMR is dependent on the size of FFT window. As shown in Fig. 2, the larger the FFT window, the higher the SNR. However, the gain yielded by enlarging FFT window gradually diminishes as the size of FFT window increases. To understand why, consider a FFT bin (whose size equals the inversion of FFT window size) that is large enough for containing one sub-clock. In this case, one can always reduce the FFT bin to suppress noise without affecting the sub-clock, thereby increasing SNR. On the other hand, if the size of FFT bin is already smaller than the bandwidth of sub-clock, then further reducing FFT bin will also reduce the energy of contained memory EMR, thus providing no SNR gain. Based on the results shown in Fig. 2, we set FFT window size to 1s in the following measurements.

Fig. 3 plots the 3 dB range of memory EMR measured from different directions. We observe that receiving direction has a slight impact on range, which should be attributed to the shape of shielding cases of particular devices. Moreover, because of the lower operating voltage, the range of DDR4-2133's EMR is about 25% shorter than that of DDR3-1600. Nevertheless, even when receiving from the worst direction, the 3dB ranges of DDR3-1600 and DDR4-2133 exceed 25m and 20m, respectively.

4.3 Response to Stimulus

To understand the impact of stimulus network traffic on eavesdropper's memory EMR, we setup an experiment where the laptop equipped with DDR4-2133 is employed to eavesdrop on an 802.11n transmitter. The experiment is conducted on a clean channel to avoid the interference of uncontrolled network traffic. The 802.11n transmitter is configured to send an 100 ms UDP flow every 200 ms. We then vary the rate of the UDP flow and repeat the experiment.

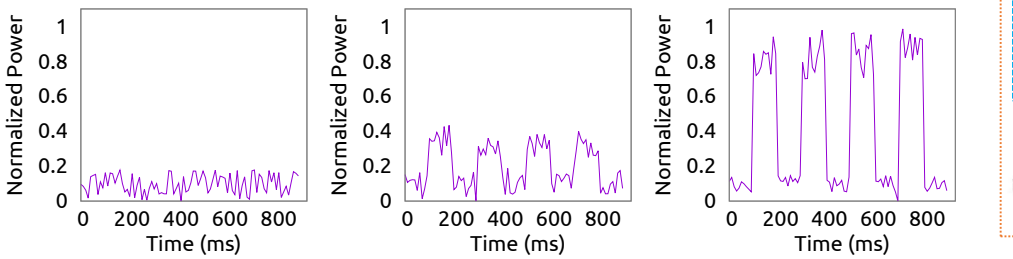
Fig. 4 shows the time varying amplitude of memory EMR measured in close proximity to the eavesdropper using a sliding FFT window of 100 ms. As shown in Fig. 4, the eavesdropper's memory EMR demonstrates a clear responsive pattern when the rate of UDP flow increases to only 2 Mbps. The amplitude of response can be significantly boosted by further increasing the rate of stimulus network traffic.

5 EarFisher Overview

EarFisher is designed as a standalone system to detect eavesdroppers in a wireless network without the cooperation of other network nodes. It differentiates eavesdroppers from legitimate receivers based on an architectural criteria, where receivers are convicted of eavesdropping as long as they transfer other devices' packets to memory. In contrast, a legitimate receivers should drop other devices' packets immediately in wireless NICs.

As illustrated in Fig. 5, EarFisher consists of a stimulator and a detector. The stimulator is a wireless network of two cooperative nodes, which exchange packets to generate stimuli[2]. The detector senses memory EMR using a software defined radio, which is hosted by one of the stimulator nodes and is synchronized with the wireless NIC to monitor the variations of memory EMRs under traffic stimuli.

[2]A simpler method is to use a single wireless device to forge a packet flow. However, crafty eavesdroppers may detect the forged packet flow by noticing that the sender and the receiver manifest the same channel state information.



(a) Rate = 600 Kbps. (b) Rate = 2 Mbps. (c) Rate = 20 Mbps.

Figure 4: Time varying amplitude of memory EMR under the stimuli of periodic UDP flows.

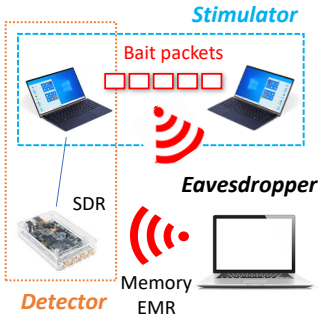


Figure 5: EarFisher architecture.

At a high-level, EarFisher features four key designs.

Sensing memory EMRs. Despite the amplification by multi-channel architecture, memory EMR is still very weak and thus can be easily buried by noise when sensing from a distance. Moreover, when devices having the same memory frequency coexist in an environment, their memory EMRs will be mixed together in frequency spectrum, making it difficult to track them separately. EarFisher addresses these challenges by using folding [39] – a signal processing algorithm originally used by large radio telescopes to amplify astronomical signals – to sense weak memory EMRs. It then leverages the fingerprint of memory clock to separate and track the memory EMRs of different devices.

Obscuring stimulus traffic. Sensing weak memory EMRs requires a large FFT window to suppress noise. However, intensive transmission of stimulus traffic in a large time window may interfere with normal network traffic. In addition, it may introduce a distinguishable traffic pattern to alert crafty eavesdroppers. To address this challenge, EarFisher first splits a large block of bait packets into small pieces and then disguises them as normal network traffic. At the detector side, EarFisher stitches signal samples captured at the time instants of stimuli into a complete window before taking FFT.

Tolerating system memory workloads. Memory workloads of operating systems and applications also produce memory EMR, which is difficult to distinguish from the response of eavesdropper when memory activities coincidentally occur at the time instants of traffic stimuli. To avoid false alarm, EarFisher profiles the memory EMR incurred by system memory workloads for each device at runtime. It then tests the hypothesis if the surge of memory EMR under stimuli is sufficiently significant to claim a detection of response.

Detecting countermeasure. Eavesdroppers knowing the design and presence of EarFisher may actively write memory to emit strong EMR, which will interfere with EarFisher's detection. To detect this countermeasure, EarFisher exploits a fundamental dilemma faced by the eavesdropper, where intermittent writing of memory leaves significant chance of exposing the response to stimuli, while consistent writing presents an abnormal EMR pattern that can be distinguished from normal system memory workloads.

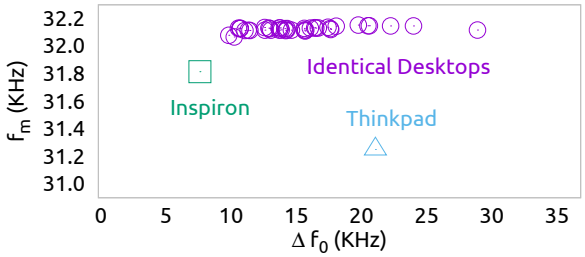


Figure 6: The distribution of memory clock fingerprints.

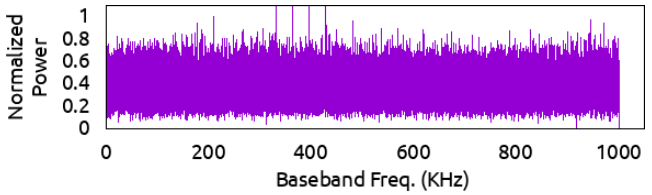
6 System Design

This section presents the design of EarFisher in detail. We first propose a sensing primitive to monitor memory EMRs, and then describe the design of stimulator and detector. Finally, we discuss how to detect eavesdroppers who deliberately write memory to interfere with EarFisher's detection.

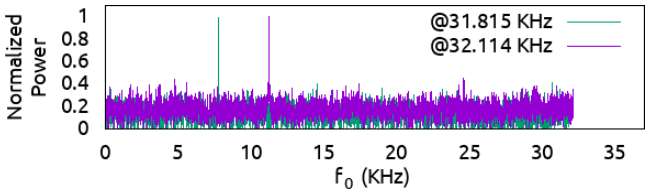
6.1 Sensing Memory EMRs

As discussed in section 4, memory EMR consists of a series of sub-clocks, where the i-th sub-clock is at $f_0 - if_m$. Due to minute manufacturing deviations, f_0 and f_m present a unique fingerprint, which distributes the sub-clocks of different devices to different frequencies. In the following, we first characterize memory clock fingerprint to study if it is sufficiently diverse to allow the separation of memory EMRs. We then discuss how to fuse sub-clocks to sense memory EMRs buried by noise, and develop a signal processing algorithm that exploits memory clock fingerprint to separate and track the memory EMRs of individual devices.

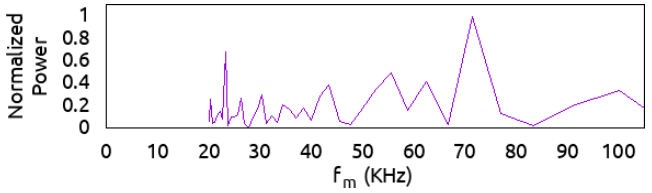
Characterizing memory clock fingerprints. To characterize the fingerprint of memory clock, we conduct measurements on 32 devices equipped with DDR3-1600, including a Dell Inspiron, a Thinkpad, and 30 identical desktops in the computer room of a university library. Fig. 6 plots the distribution of their memory clock fingerprints. We observe that f_0 differs even across the 30 identical desktops. In comparison, the offset of f_m is only significant across different devices, which should be attributed to the different modulation frequencies of their clock generators.



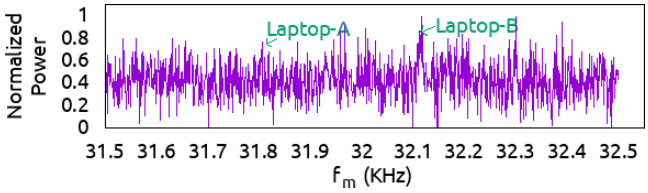
(a) Noisy spectrum after FFT.



(b) Results of folding.



(c) FFT.



(d) Auto-correlation.

Figure 7: Comparison between folding, auto-correlation and FFT over a noisy spectrum containing the memory EMRs of two computers whose f_m are 31.815 KHz and 32.114 KHz, respectively.

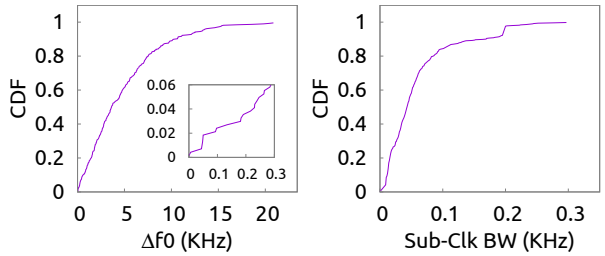


Figure 8: The CDFs of Δf_0 and sub-clock bandwidth.

To validate if the fingerprints are sufficiently diverse, consider an arbitrary pair of identical desktops. As they have similar f_m, the separation of their sub-clock components solely depends on the offset of f_0. Specifically, if the offset of f_0 is too small, sub-clock components will overlap in frequency in a pair-wise fashion. In this case, the overlapped bandwidth can be computed as,

$$\Delta BW = \frac{BW_a + BW_b}{2} - \Delta f_0,$$

where Δf_0 is the offset of f_0; BW_a and BW_b are the sub-clock bandwidth of the two devices, respectively. Clearly, their memory EMRs are separable if and only if $\frac{BW_a+BW_b}{2}$ is smaller than Δf_0. Fig. 8 further compares the distributions of Δf_0 and $\frac{BW_a+BW_b}{2}$ measured on the 30 identical desktops. We observe that Δf_0 is significantly larger than sub-clock bandwidth. Specifically, Δf_0 is larger than 300 Hz in 94% cases, whereas the bandwidth of all sub-clocks are smaller than 300 Hz.

Based on f_0 and sub-clock bandwidth measured on the 30 identical desktops, we further conduct a simulation to test the capacity of a memory 'channel', i.e., the maximum number of identical devices that can coexist on the same memory frequency without mixing memory EMRs. In each run of the simulation, we randomly add desktops to the memory channel until the produced memory EMRs become inseparable. After 10000 runs, we find that the average capacity is 7. We note that the capacity should be significantly higher if coexisting devices are different. In particular, for a pair of different devices, even if some of their sub-clock components are overlapping, others are likely separated because of different f_m.

Fusing sub-clocks. We then discuss how to fuse sub-clocks to boost the SNR of memory EMR. A key design requirement is to achieve computational efficiency, because the signal processing targets at high-resolution frequency spectra obtained by large FFT windows.

To this end, we propose a novel use of *folding* – a fast algorithm originally used by large radio telescopes to amplify periodic astronomical signals [23, 31, 39]. EarFisher utilizes folding to search for sub-clock components distributed over frequency. Suppose $\mathcal{P}$ represents the series of N points of the spectrum and $\mathcal{P}[i]$ ($i \in [1,N]$ is the amplitude of the ith point. The objective of folding is to search for a signal with a period of T. The spectrum is first divided into small windows of T points and then added in a window-wise fashion as,

$$\mathcal{F}_T[i] = \sum_{j=0}^{\lfloor \frac{N}{T} \rfloor - 1} \mathcal{P}[i + j * T].$$

When folding up the spectrum using a window size of f_m, the energies of sub-clock components will be fused while the sum of noise is likely smaller due to their non-periodicity. The position of *folding peak*, i.e., the i that maximizes $\|\mathcal{F}_T[i]\|$, is dependent on the offset between receiving frequency and the memory clock's f_0. Because the f_m of memory clock is unknown, EarFisher performs folding at each possible f_m to search for memory EMRs. Fig. 7a shows an example of a noisy spectrum containing the memory EMRs of two laptops, whose f_m is 31.815 KHz and 32.114 KHz, respectively. Fig. 7b plots the folded spectra where the peaks corresponding to the fused energies of sub-clocks can be clearly identified. In comparison, as shown in Fig. 7c and Fig. 7d, the performance of auto-correlation and FFT – two widely used signal pro-

cessing algorithm of periodic signal detection – is worse than folding despite their higher computational overhead. Specifically, FFT fails to identify memory EMRs due to its poor resolution. Auto-correlation identified one of the laptops but is significantly more susceptible to noise than folding.

Separating memory EMRs. EarFisher exploits the diversity of memory clock fingerprints to separate and track memory EMRs. To this end, EarFisher performs two steps of processing iteratively. First, it folds up the spectrum at all possible f_m, and then identifies the highest folding peak caused by the device that has the strongest memory EMR. Name this device as Alice. Second, EarFisher outputs the highest folding peak, which reflects the fused amplitude of Alice's memory EMR, and then removes the sub-clock components of Alice from the spectrum. The goal is to eliminate possible peaks yielded by Alice in subsequent rounds of folding, which may prevent EarFisher from identifying the folding peaks of other devices. Note that each sub-clock component may include multiple spectral points depending on its bandwidth. EarFisher identifies sub-clock bandwidth using standard edge detection algorithm [19], and then removes all points included in the peak. The above procedure is repeated until the highest folding peak falls below a predefined threshold.

In practice, the spectrum may contain other signals produced by wireless communication. EarFisher classifies memory EMR and wireless communication based on two simple rules. First, the highest folding peak should trace-back to at least two peaks separated by the folding period in the spectrum. Second, the bandwidth of each trace-backed peak should not exceed 300 Hz – an empirical upper bound on sub-clock bandwidth obtained through extensive measurements. In comparison, the signal bandwidth of wireless communication is typically orders-of-magnitude higher in order to achieve a meaningful data rate.

Tracking memory EMRs. EarFisher tracks memory EMRs of different devices by using (f_0, f_m) as a device identity. In practice, (f_0, f_m) may experience small variance over time. To address this issue, EarFisher clusters folding peaks obtained at different time instants based on the euclidean distance of (f_0, f_m), where each cluster corresponds to one device. It then assigns a unique ID to each device and tracks the variation of f_0 and f_m using standard phase-locked loops.

6.2 Stimulator

The stimulator of EarFisher consists of two cooperative devices, which exchange packets to generate stimulus traffic. To detect eavesdroppers in a specific wireless network, the bait packets should be transmitted on the same frequency channel. In case the network to protect is operated on multiple channels, the stimulator can hop across channels to inject baits. In the following, we focus on the design of a Wi-Fi stimulator. The principle of the presented design is broadly applicable to other types of wireless networks.

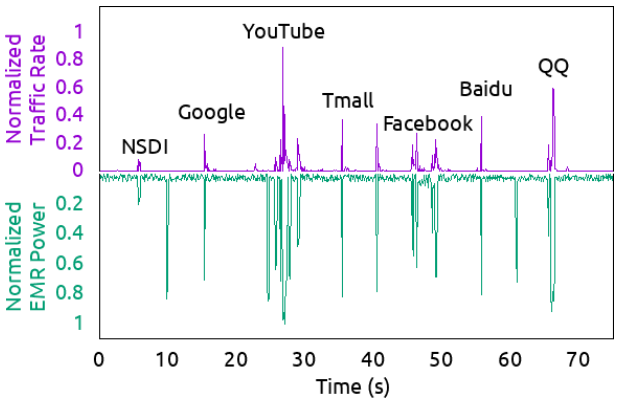


Figure 9: Eavesdropper's response to the stimuli of web page downloads.

Engineering stimulus traffic. EarFisher disguises the network of Wi-Fi stimulator as a WLAN where one device hosts a virtual access point and the other is attached as a client. To stimulate a Wi-Fi eavesdropper without incurring its alert, the client launches a sequence of webpage downloads at random time instants, where each download generates a short stimulus consisting of several MBs of data depending on the size of webpage[3]. Fig. 9 shows an eavesdropper's response to stimuli when it is sniffing on the downloads of the homepage of NSDI as well as 6 popular pages top-ranked in Alexa [2]. We observe that YouTube triggered the strongest response due to its large page size, suggesting that the stimulator should leverage media-rich pages for stimuli.

In practice, round-trip delays occurred at upper-layers prevent the stimulator from achieving a high throughput, thereby degrading the intensity of stimulus. For example, as can be seen in Fig. 9, the downloading traffic of Facebook page was divided into two parts due to large upper-layer delay, which significantly weakens the eavesdropper's response despite the large size of Facebook page. To address this issue, EarFisher first records the real traffic of webpage downloads, and then replays the traffic in the local network of stimulator.

Media access control. To further improve the effectiveness of stimulus, EarFisher leverages the following MAC-layer schemes. First, EarFisher utilizes the frame aggregation feature of 802.11 to bundle multiple bait packets in a single transmission, which effectively increases the throughput of stimulus traffic. Second, before transmitting bait packets, EarFisher uses RTS and CTS to mute normal network traffic. The goal is to reduce the chance of false alarm, which may happen if legitimate network nodes transmit or receive and thereby produce memory EMRs during stimulus period. One of our future work is to further control the timing of stimuli to minimize the interference with normal network traffic. This can be achieved by predicting the variation of normal network traffic using theoretical models [28] and then sending stimulus traffic only when the wireless channel is under-utilized.

[3] According to httparchive [4], the average webpage size has increased from 1.6 MB in 2014 to 4 MB in 2019.

6.3 Detector

At a high-level, the detector of EarFisher separates and tracks memory EMRs using the sensing primitive proposed in section 6.1, and then inspects each memory EMR to infer the response to traffic stimuli. Note that whenever transmitting bait packets, the stimulator will also emit memory EMR, which is difficult to distinguish from the response of an eavesdropper. To address this issue, a straightforward method is to count bait receivers and check if the number is larger than expected. In the design of EarFisher, we employ a simple alternative to walkaround receiver counting. Specifically, EarFisher employs two stimulators of different memory frequencies. In this case, because the eavesdropper's memory frequency must differ from that of at least one stimulator, the detector can quickly identify eavesdropper's memory EMRs based simply on emission frequency.

Profiling memory EMRs. To detect the presence of eavesdroppers, EarFisher compares the amplitudes of memory EMRs measured in the presence and absence of stimulus traffic, named as *stimulus set* and *baseline set*, respectively.

To profile the stimulus set, EarFisher first captures signals around the transmission time of bait packets, and then puts captured signals in a large FFT window. Once the window is completely filled, EarFisher runs the algorithm proposed in section 6.1 to identify memory EMRs. Suppose n memory EMRs are observed at $I^i = (f_0^i, f_m^i)$, $i \in (1,n)$. For each I^i, EarFisher establishes a stimulus set to record the amplitude of memory EMR, and then populates the set when new FFT windows are available.

Note that a surge of memory EMR under traffic stimuli could be a coincidence caused by system memory workload. To profile the probability of such coincidence, EarFisher keeps tracking the amplitude of memory EMR for each I^i to build the baseline set. To prevent memory EMRs caused by network activity from polluting the baseline set, EarFisher purges signals captured in the presence of ongoing network traffic, and then stitches remaining signals into FFT windows.

Statistical hypothesis testing. EarFisher then compares the stimulus and baseline set using a statistical hypothesis test called t-test, which is widely used to decide whether a drug has had a significant effect on the studied population. A t-test takes the means, variances, and the number of samples of the two compared sets, and then computes a t-value as follows,

$$t = \frac{\mu_{\text{stimulus}} - \mu_{\text{reference}}}{\sqrt{\frac{\sigma_{\text{stimulus}}}{n_{\text{stimulus}}} + \frac{\sigma_{\text{reference}}}{n_{\text{reference}}}}}, \tag{3}$$

where μ and σ are the mean and variance of a set, respectively; n is the number of samples, which determines the degrees of freedom of the test.

Once the t-value and degrees of freedom are determined, a p-value can be calculated. A large positive p-value is an evidence that μ_{stimulus} is significantly larger than μ_{ref}. EarFisher then compares the p-value with a a chosen level of statistical significance, denoted as α. Basically, a high α assures low false alarm rate but may raise miss detection. EarFisher exposes the configuration of α, which allows users to tune the balance between detection rate and false alarm.

6.4 Detecting Countermeasure

To counteract the detection of EarFisher, eavesdroppers knowing the design and presence of EarFisher may actively write memory to emit strong EMR, which will pollute the baseline set profiled by EarFisher, thus defeating the statistical test given in Eqn. 3. In the following, we propose a simple method to detect this countermeasure.

The dilemma of eavesdropper. Our insight is that, when actively writing memory, the eavesdropper faces a fundamental dilemma where intermittent writing leaves significant chance of exposing the response to stimuli, while consistent writing presents an abnormal pattern of memory EMR that can be distinguished from normal system memory workloads.

Specifically, to mask the response to traffic stimuli, the eavesdropper must erase the difference between the reference and stimulus set. However, because the stimulus set is built by stitching signals received under traffic stimuli, it captures the eavesdropper's memory EMR in a status of virtually consistent writing of memory. In order to defeat the statistical test, the eavesdropper must write memory at a comparable intensity to corrupt the baseline set.

On the other hand, due to the presence of hierarchical cache, a normal system rarely writes memory consistently. In particular, unlike wireless NICs that write received packets directly into memory, legitimate programs only read/write memory under cache miss, while modern Intel and AMD CPUs can maintain cache hit rate above 90%. Moreover, due to the high bus bandwidth of modern memory (ranging from a couple of GB/s of DDR to tens of GB/s of DDR4), the data transfer caused by normal memory workload typically completes in very short time, producing only intermittent bursts of memory EMRs. As an example, Fig. 10 shows the time-varying amplitude of memory EMR measured on a laptop running Ubuntu 18.04 and Windows 10 with no other programs. We observe that the operating systems alone yield noticeable variance of memory EMR. In contrast, consistent writing of memory can easily erase the variance, presenting a distinguishable pattern.

Exploiting the dilemma. To detect eavesdroppers who deliberately write memory to mask response to traffic stimuli, EarFisher complements the detector proposed in section 6.3 with an auxiliary detector to inspect abnormal EMR pattern caused by deliberate writing.

The auxiliary detector uses the variance of normalized memory EMR amplitude as a feature to investigate if the baseline set has been polluted by deliberate writing. However, because measuring memory EMR requires a large FFT window to suppress noise, obtaining a fine-grained estimation of EMR

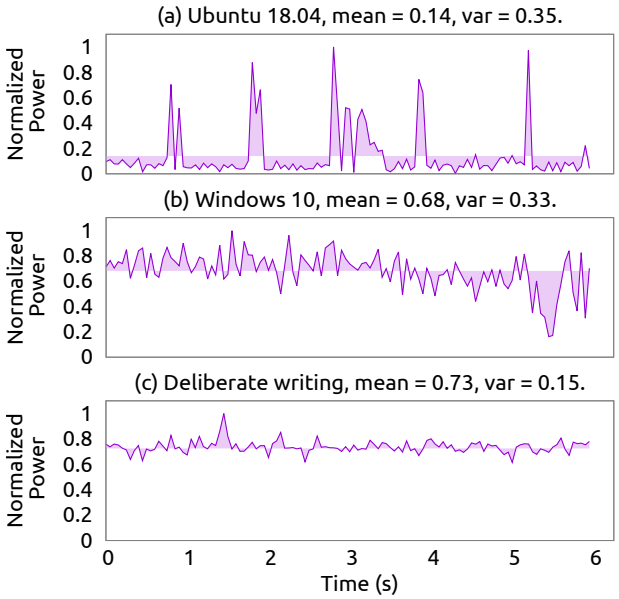


Figure 10: The time varying memory EMR induced by operating systems and deliberate writing.

variance may incur a long delay. EarFisher circumvents this issue by *oversampling* the baseline set. Specifically, EarFisher divides each FFT window in the baseline set into small blocks of signals, and then randomly picks blocks from different FFT windows to create new EMR samples. EarFisher then calculates the variance of created EMR samples, and repeats this for multiple rounds to profile a fine-grained distribution of EMR variance using a small number of FFT windows.

Once the distribution of EMR variance has been profiled, EarFisher performs t-test again to check if the mean of the distribution is smaller than an empirical threshold. To determine the threshold, we conduct extensive measurements on devices of different configurations. Our measurements lead to several findings. First, the variance is smaller on devices of larger cache and faster memory. Second, the variance is minimum when a devices is running no program except the operating system. Third, Windows typically demonstrates smaller variance than other operating systems. In our measurements condcuted on 39 devices, the minimum and maximum variance are 0.32 and 0.35, which are measured on a Dell Inspiron equipped with an 8 MB cache and DDR4-2133 running Windows 10, and a Thinkpad equipped with a 4 MB cache and DDR3-1600 running Ubuntu 18.04, respectively. Notice that the measured EMR variance may vary under different SNRs, we further profile the minimum EMR variance on Inspiron running Windows under different attenuation conditions. Before taking statistical test, EarFisher chooses a threshold based on the measured SNR of memory EMR. To avoid false alarm and account for devices of higher configurations, the current design of the auxiliary detector adopts a conservative threshold that is 10% lower than the empirically profiled minimum variance. Further refining the threshold for higher-end cache and memory configurations is left to our future work.

7 Evaluation

This section evaluates the performance of EarFisher in an 802.11n network. The current prototype of EarFisher employs BladeRF [3] to sense memory EMRs. To generate stimulus traffic, EarFisher replays the recorded traffic of YouTube page download at about 10 MB/s in its local stimulator network. Based on empirical measurements shown in Fig. 2, the FFT window size of memory EMR sensor is set to 1s, as larger windows increase detection latency but yield limited SNR gain. When performing statistical test to detect eavesdroppers, the stimulus and baseline set are profiled based on 1s and 3s of memory EMR signals, respectively. To improve the granularity of profiling, the baseline set is oversampled to create 10 FFT windows using the method described in section 6.4.

In the following, we first evaluate the accuracy of EarFisher in detecting eavesdropper and countermeasure, and then study its performance in a real deployment scenario where three EarFisher nodes are deployed to monitor an indoor environment of about 1600 ft^2.

7.1 Eavesdropper Detection

We conduct experiments on two commodity laptops, including a Thinkpad with DDR3-1600 and a Dell Inspiron with DDR4-2133. To evaluate the detection performance of EarFisher, we let the laptops act as eavesdropper (EV) and legitimate receiver (LR), and then compare the p-values computed by EarFisher. It is important to note that the p-values of EV and LR are NOT equivalent to the probability of detection and false alarm. Instead, the final detection result depends on the choose of p-value threshold, which EarFisher exposes to the user for configurable trade-off between detection rate and false alarm. In the following, we study the impacts of four key factors on EarFisher's detection performance, including the attenuation of memory EMR, system memory workloads, normal network traffic, and the interfering EMRs emitted by coexisting devices that have the same memory frequency.

Attenuation. We first study the impact of EMR attenuation on EarFisher's detection performance. We note that accurately controling EMR attenuation is difficult because we cannot connect an attenuator to the 'antenna' of the EMR emitter, i.e., the memory bus. To walk-around this issue, we first record the memory EMR of eavesdropper in close proximity, and then emulate a certain level of attenuation by mixing the recorded signal with an equivalent amount of white noise. In this experiment, the eavesdropper runs no software except OS, which allows us to exclude the interference of system memory workloads and study the optimal detection performance of EarFisher as a function of EMR attenuation.

As shown in Fig. 11, we observe that the p-values of DDR3 and DDR4 eavesdroppers are consistently higher than 0.9 before the amount of attenuation exceeds 29 dB and 21 dB, which typically translate to a line-of-sight path loss of about

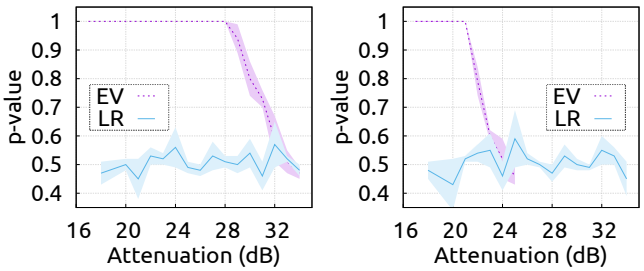
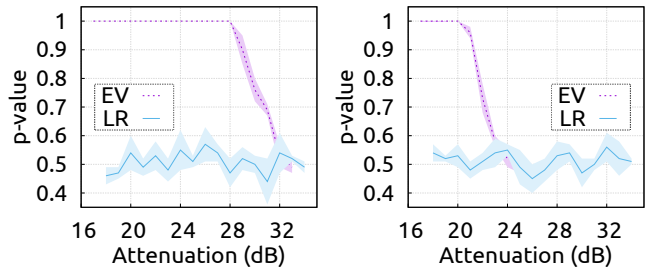
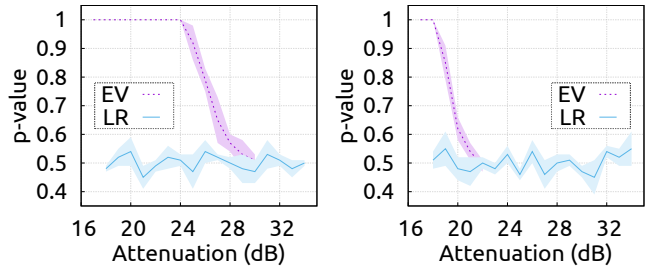


Figure 11: The p-values for the eavesdropper (EV) and the legitimate receiver (LR) as a function of EMR attenuation (left: DDR3, right: DDR4).



(a) Wireshark.



(b) VLC media player.

Figure 12: The impact of memory workloads (left: DDR3, right: DDR4).

30 m and 24 m, respectively. When the amount of attenuation further increases, the p-value for eavesdropper begins to decrease, as the surge of eavesdropper's memory EMR corresponding to the response to stimuli is gradually submerged by noise. We note that DDR3 is more resistant to attenuation than DDR4 because of the stronger EMR attributed to the higher operating voltage. In comparison, the p-value of legitimate receivers fluctuates in between 0.4 to 0.6 consistently despite the increase of attenuation. The results indicate that, if the user chooses a p-value threshold of 0.9, then EarFisher will detect all DDR3 and DDR4 eavesdroppers before attenuation reaches 29 dB and 21 dB while without miss-classifying any legitimate receivers as eavesdroppers.

System memory workload. Memory EMR produced by system memory workload will pollute the baseline profiled by EarFisher and thus will affect the result of statistical test. To study the impact of system memory workload, we let the laptops run two representative applications, including Wireshark – a widely used packet analyzer, and VLC media player, which reads memory intensively to load a high-definition video.

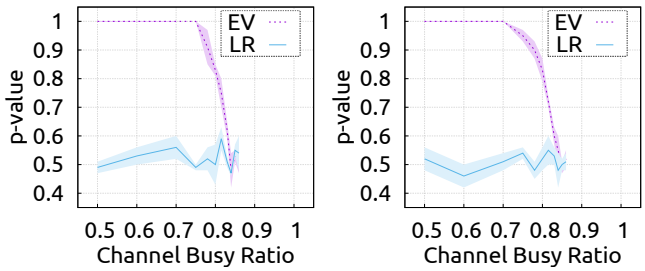


Figure 13: The impact of network traffic (left:DDR3, right:DDR4).

As shown in Fig. 12, compared with the results shown in Fig. 11, the attenuation resistance of EarFisher degrades by less than 1 dB and 5 dB when the eavesdroppers run Wireshark and VLC, respectively. VLC imposes a higher impact because it involves more frequent memory activities and thus produces a higher level of pollution to the baseline set. Nevertheless, even under the interference of VLC, the p-values of the DDR3 and DDR4 eavesdroppers are consistently higher than that of legitimate receivers as long as the levels of attenuation are below 25 dB and 19 dB, respectively.

We also observe that the p-values of legitimate receivers are not affected by Wireshark and VLC on both DDR3 and DDR4 laptops. This is because system memory workload will impact the baseline and stimulus set uniformly, hence the results of statistical test will remain unbiased.

Although we focus on only two representative applications in this experiment, we note that the interference caused by memory workload can be generally quantified using the busy ratio of memory bus. We will study EarFisher's performance as a function of memory busy ratio in section 7.2.

Network traffic. To study the impact of network traffic, we employ two additional 802.11n nodes to inject normal network traffic using iPerf [5]. In this experiment, we place eavesdroppers at 10m away from the EarFisher's detector. We then compare the p-values for eavesdroppers and legitimate receivers in the presence of different volumes of network traffic. We use channel busy ratio to quantify the interference produced by normal network traffic, because absolute volume can be misleading when characterizing interference intensity in wireless networks of different data rates.

As shown in Fig. 13, we observe that the detection performance of EarFisher is reliable as long as the channel busy ratio is below 73%. As channel busy ratio further increases, it becomes increasingly difficult to profile a clean baseline set not affected by network traffic. As a result, the p-value of the eavesdropper will begin to degrade.

Coexisting devices. We next evaluate EarFisher in a crowded environment where devices having the same memory frequency introduce interfering memory EMRs. We place the DDR3 eavesdropper in a library computer room that has 40 identical desktops, all having the same memory frequency as the eavesdropper. We then turn on desktops one by one and study the impact on EarFisher's performance.

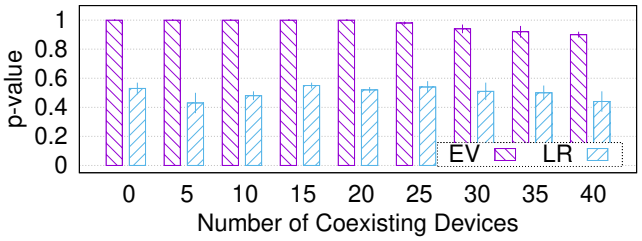


Figure 14: The impact of coexisting devices.

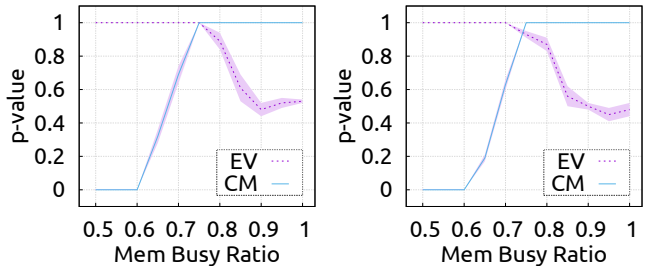


Figure 15: The p-values of the eavesdropper (EV) and the countermeasure (CM) detectors under different memory busy ratios (left: DDR3, right: DDR4).

Fig. 14 compares the p-values for eavesdropper and legitimate receiver as the number of coexisting desktops increases. We observe that EarFisher is fairly resistant to the interference of coexisting devices. Even when all 40 desktops are active simultaneously, the p-value for eavesdropper remains higher than 0.8, while the p-value of legitimate receiver fluctuates around 0.5. This should be attributed to the diversity of memory clock fingerprint, which allows EarFisher to separate the memory EMRs of different devices.

7.2 Countermeasure Detection

We next evaluate EarFisher's performance of detecting eavesdroppers that deliberately write memory to mask response to stimuli. We explore the parameter space of eavesdropper's countermeasure by tuning the ratio of deliberate writing, which results in different memory busy ratios. We then study the impacts on the p-values calculated by EarFisher's eavesdropper and countermeasure detector, denoted as EV and CM in Fig. 15, respectively. In this experiment, the eavesdropper is placed at 12 m away from EarFisher, which causes a path loss of about 15 dB.

As shown in Fig. 15, we observe that, as memory busy ratio increases, the p-value of eavesdropper detector decreases because the baseline set becomes increasingly polluted. In contrast, the p-value of countermeasure increases because the baseline set demonstrates increasing abnormality. For example, as the p-value for the DDR4 eavesdropper begins to drop when memory busy ratio increases to above 0.75, the p-value of countermeasure detector has reached 1.0. The results validate that the eavesdropper cannot defeat EarFisher's eavesdropper and countermeasure detectors at the same time.

7.3 A Deployment Case

In the following, we evaluate EarFisher on a testbed where three EarFisher nodes are deployed at different locations (i.e., S1-S3) in an indoor environment of 1600 ft^2, as shown in Fig. 16. We place the eavesdroppers at 9 locations in different rooms. We then randomly choose a node at another deployment location to stream a live video. The eavesdropper runs Wireshark and sniffs on the streamed video. All doors are closed during experiment. We study the precision and recall of EarFisher when the p-value threshold is set to 0.6 and 0.75, respectively.

As shown in Fig. 17, when the p-value threshold is set to 0.75, 8 of 9 DDR3 and 7 of 9 DDR4 eavesdroppers can be detected by at least one of the three EarFisher nodes with a recall higher than 90% while incurring no false alarms. In particular, eavesdroppers at location B and C can be accurately detected by EarFisher deployed at S3 and S2, despite the block of doors and walls. When the p-value threshold is relaxed to 0.6, the recall for the DDR4 eavesdropper at location F increases from 0% to 83%. However, this is at the cost of slightly reducing the precision from 100% to 94% and 97% at location A and B, respectively. We find that location G demonstrates as a blind spot due to severe EMR attenuation. We note that the performance of EarFisher can be further improved by leveraging advanced radio equipment such as high-gain LNA, better planing the deployment of EarFisher, or deploying more EarFisher nodes.

8 Discussion

In this section, we discuss important issues related to the design of EarFisher, including response mitigation-based countermeasures and EarFisher's limitations.

8.1 Response Mitigation

As discussed in section 6.3, EarFisher detects eavesdroppers by comparing the baseline and stimulus set to identify eavesdroppers' responses to stimuli. Accordingly, eavesdroppers can counter EarFisher by either polluting the baseline or mitigating the evidence of response in the stimulus set. In section 6.4 and section 7.2, we have shown how EarFisher effectively detects the first countermeasure. In the following, we discuss and analyze the second.

Eavesdropping on specified receivers. By modifying the firmware of wireless NIC, eavesdroppers can be configured to only sniff on packets transmitted to a specified receiver. EarFisher can detect such attacks by counting the receivers of packets sent to a specific address. This is feasible as long as the memory EMRs of the eavesdropper and the legitimate receiver can be separated in frequency spectrum, which is at a high probability as measured and discussed in section 6.1.

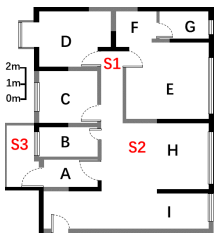


Figure 16: Testbed deployment.

	DDR3, p-value threshold = 0.75						DDR4, p-value threshold = 0.75						DDR4, p-value threshold = 0.60					
	S1		S2		S3		S1		S2		S3		S1		S2		S3	
	Rc	Pr	Rc	Pr	Rc	Pr	Rc	Pr	Rc	Pr	Rc	Pr	Rc	Pr	Rc	Pr	Rc	Pr
A	0	0	0.34	1.00	1.00	1.00	0	0	0	0	1.00	1.00	0	0	0.52	1.00	1.00	0.94
B	0	0	0.05	1.00	1.00	1.00	0	0	0	0	0.93	1.00	0	0	0.07	1.00	1.00	0.97
C	1.00	1.00	1.00	1.00	0	0	0	0	0.90	1.00	0	0	0.63	1.00	1.00	1.00	0	0
D	1.00	1.00	0	0	0	0	1.00	1.00	0	0	0	0	1.00	1.00	0	0	0	0
E	1.00	1.00	0.97	1.00	0	0	1.00	1.00	0	0	0	0	1.00	1.00	0	0	0	0
F	1.00	1.00	0	0	0	0	0	0	0	0	0	0	0.83	0.96	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0.57	1.00	1.00	1.00	0	0	0	0	1.00	1.00	0	0	0.61	1.00	1.00	1.00	0	0
I	0.12	1.00	1.00	1.00	0	0	0	0	1.00	1.00	0	0	0	0	1.00	1.00	0	0

Figure 17: The recall (Re) and precision (Pr) for eavesdroppers deployed at different locations.

Weaponizing low-power wireless. Eavesdroppers may circumvent EarFisher's sensing by repurposing low-power wireless devices. As an example, ESP8266 [33] – a Wi-Fi enabled IoT – has a microcontroller that directly controls Wi-Fi chip via UART, thus avoiding strong memory EMR when receiving packets. However, such low-power architectural features typically limit the computation and storage capability of IoTs, making them ill-suited for security-intrusive tasks. For example, the on-board memory of ESP8266 is only 32 KiB for instruction and 80 KiB for data. In comparison, to crack a WEP key, the attacker needs to perform a computation over millions of encrypted packets [14]. As a result, to work as a full-fledged eavesdropper, low-power wireless devices have to rely on a host, which will leak strong memory EMR that can be captured by EarFisher.

Physical shielding. A physical method to mitigate response is to shield eavesdropper's memory bus to attenuate EMR. However, unlike shielding external emanation sources such as monitor cables, shielding memory bus can be prohibitively challenging and expensive.

8.2 Limitations

Excessive verdicts. By EarFisher's detection methodology, any device that digests others' packets in CPU-memory system will be convicted of eavesdropping. Unfortunately, it is difficult, if not impossible, to differentiate benign or malicious use of other's packets. As a result, all software radios will be identified as eavesdroppers as long as they transfer baseband signals to a host or process signals on board, both will emit strong memory EMRs. This is harsh but reasonable, because software radios process other devices' signal (albeit such processing may be only at the PHY layer) in an intrusted context with rich storage and computational resources capable of security- and privacy-intrusive tasks.

Besides software radios, recent wireless sensing and communication primitives such as backscatter [20] and localization [6, 38], may require Wi-Fi NICs to operate in monitor mode. To authenticate these applications, a possible method is to register legal eavesdropping devices a priori, and then let EarFisher count the number of eavesdroppers to determine the presence of illegitimate ones.

Low rate wireless networks. Experiment results shown in Fig. 4 suggest that a traffic stimulus of as slow as 2 MB/s suffice to trigger the eavesdropper's response. Unfortunately, this is still beyond the maximum data rate of many low-power wireless networks such as ZigBee. However, we expect that EarFisher will achieve better detection performance in the coming generation of high-rate wireless networks such as IEEE 802.11ax, which features GB/s level data rate, thus allowing for traffic stimuli of much higher intensities.

Blind spots. Eavesdroppers knowing the deployment of EarFisher may exploit locations subject to severe EM attenuation, such as room G shown in Fig. 16. Another example is to deploy eavesdroppers as *hidden terminals*, where the eavesdropper can hear the packets of a transmitter-of-interest, but is at a location relatively distant to EarFisher, such that the memory EMR cannot be accurately sensed. Such blind spots of detection can be mitigated by extending the coverage of EarFisher. Possible methods include but not limited to using high-gain LNA, leveraging advanced signal processing such as blind beamforming [12], or deploying more EarFisher nodes to monitor the area-of-interest.

9 Conclusion

This paper presents EarFisher – a system that detects wireless eavesdroppers by stimulating and sensing memory EMRs. Experiment results show that EarFisher accurately detects eavesdroppers despite poor signal conditions and the interference of normal network traffic, system memory workloads, and the interfering EMRs emitted by coexisting devices. We believe EarFisher provides an important block for building secure wireless networks. Incorporating EarFisher in wireless security protocols, such as to verify the confidentiality of key establishment, remains an important problem for future work.

Acknowledgments

We are grateful to NSDI reviewers and our shepherd, Andreas Haeberlen, for their insightful comments. This research was supported, in part, by funds from BvTech S.p.A. and the members of the Cybersecurity at MIT Sloan (CAMS) consortium (https://cams.mit.edu)

References

[1] Aircrack-ng. https://aircrack-ng.org.

[2] Alexa topsites. https://www.alexa.com/topsites.

[3] Bladerf. https://www.nuand.com.

[4] httparchive. https://httparchive.org/.

[5] Iperf. https://iperf.fr/.

[6] Fadel Adib, Zachary Kabelac, and Dina Katabi. Multi-person localization via rf body reflections. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[7] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The em side-channel(s). In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2002.

[8] Monjur Alam, Haider Adnan Khan, Moumita Dey, Nishith Sinha, Robert Callan, Alenka Zajic, and Milos Prvulovic. One&done: A single-decryption em-based attack on openssl's constant-time blinded rsa. In *USENIX Security Symposium*, 2018.

[9] Charles Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. In *The International Conference on Computers, Systems and Signal Processing*, 1984.

[10] Boualem Boashash. Time-frequency signal analysis and processing: A comprehensive reference. 2003.

[11] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurelien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[12] J. F. Cardoso and A. Souloumiac. Blind beamforming for non-gaussian signals. *IEE Proceedings F - Radar and Signal Processing*, 1993.

[13] Anadi Chaman, Jiaming Wang, Jiachen Sun, Haitham Hassanieh, and Romit Roy Choudhury. Ghostbuster: Detecting the presence of hidden eavesdroppers. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2018.

[14] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of rc4. In *Annual International Workshop on Selected Areas in Cryptography*, 2001.

[15] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2001.

[16] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom. Ecdsa key extraction from mobile devices via nonintrusive physical side channels. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.

[17] Mordechai Guri, Assaf Kachlon, Ofer Hasson, Gabi Kedma, Yisroel Mirsky, and Yuval Elovici. Gsmem: Data exfiltration from air-gapped computers over gsm frequencies. In *USENIX Security Symposium*, 2015.

[18] Yi Han, Sriharsha Etigowni, Hua Liu, Saman Zonouz, and Athina Petropulu. Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[19] R. M. Haralick. Digital step edges from zero crossing of second directional derivatives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1984.

[20] Vikram Iyer, Vamsi Talla, Bryce Kellogg, Shyamnath Gollakota, and Joshua Smith. Inter-technology backscatter: Towards internet connectivity for implanted devices. In *Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2016.

[21] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *International Cryptology Conference (CRYPTO)*, 1999.

[22] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *International Cryptology Conference (CRYPTO)*, 1996.

[23] RVE Lovelace, JM Sutton, and EE Salpeter. Digital search methods for pulsars. *Nature*, 1969.

[24] Amitav Mukherjee and A Lee Swindlehurst. Detecting passive eavesdroppers in the mimo wiretap channel. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

[25] Alireza Nazari, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. Eddie: Em-based detection of deviations in program execution. In *The International Symposium on Computer Architecture (ISCA)*, 2017.

[26] Sanghoon Park, Lawrence E Larson, and Laurence B Milstein. Hidden mobile terminal device discovery in a uwb environment. In *IEEE International Conference on Ultra-Wideband*, 2006.

[27] Sanghoon Park, Lawrence E Larson, and Laurence B Milstein. An rf receiver detection technique for cognitive radio coexistence. *IEEE Transactions on Circuits and Systems*, 2010.

[28] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 1995.

[29] Nader Sehatbakhsh, Alireza Nazari, Haider Khan, Alenka Zajic, and Milos Prvulovic. Emma: Hardware/software attestation framework for embedded systems using electromagnetic signals. In *IEEE/ACM International Symposium on Microarchitecture (Micro)*, 2019.

[30] Nader Sehatbakhsh, Alireza Nazari, Alenka Zajic, and Milos Prvulovic. Spectral profiling: Observer-effect-free profiling by monitoring em emanations. In *IEEE/ACM International Symposium on Microarchitecture (Micro)*, 2016.

[31] David H Staelin. Fast folding algorithm for detection of periodic pulse trains. *Proceedings of the IEEE*, 1969.

[32] Colin Stagner, Andrew Conrad, Christopher Osterwise, Daryl G Beetner, and Steven Grant. A practical superheterodyne-receiver detector using stimulated emissions. *IEEE Transactions on Instrumentation and Measurement*, 2011.

[33] Expressif Systems. Esp8266 overview. https://www.espressif.com/products/socs/esp8266/.

[34] Vivek Thotla, Mohammad Tayeb Ahmad Ghasr, Maciej J Zawodniok, Sarangapani Jagannathan, and Sanjeev Agarwal. Detection of super-regenerative receivers using hurst parameter. *IEEE Transactions on Instrumentation and Measurement*, 2013.

[35] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in wpa2. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[36] Ben Wild and Kannan Ramchandran. Detecting primary receivers for cognitive radio applications. In *The IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2005.

[37] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Bo Li, Peter Volgyesi, and Xenofon Koutsoukos. Leveraging em side-channel information to detect rowhammer attacks. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[38] Yue Zheng, Yi Zhang, Kun Qian, Guidong Zhang, Yunhao Liu, Chenshu Wu, and Zheng Yang. Zero-effort cross-domain gesture recognition with wi-fi. In *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2019.

[39] Ruogu Zhou, Yongping Xiong, Guoliang Xing, Limin Sun, and Jian Ma. Zifi: wireless lan discovery via zigbee interference signatures. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2010.