Synesthesia: Detecting Screen Content via Remote Acoustic Side Channels*

Daniel Genkin
University of Michigan
genkin@umich.edu

Mihir Pattani
University of Pennsylvania
mihirsa@seas.upenn.edu

Roei Schuster
Tel Aviv University, Cornell Tech
rs864@cornell.edu

Eran Tromer
Tel Aviv University, Columbia University
tromer@cs.tau.ac.il

August 21, 2018

Abstract

We show that subtle acoustic noises emanating from within computer screens can be used to detect the content displayed on the screens. This sound can be picked up by ordinary microphones built into webcams or screens, and is inadvertently transmitted to other parties, e.g., during a videoconference call or archived recordings. It can also be recorded by a smartphone or "smart speaker" placed on a desk next to the screen, or from as far as 10 meters away using a parabolic microphone.

Empirically demonstrating various attack scenarios, we show how this channel can be used for real-time detection of on-screen text, or users' input into on-screen virtual keyboards. We also demonstrate how an attacker can analyze the audio received during video call (e.g., on Google Hangout) to infer whether the other side is browsing the web in lieu of watching the video call, and which web site is displayed on their screen.

1 Introduction

Physical side-channel attacks extract information from computing systems by measuring unintended effects of a system on its physical environment. They have been used to violate the security of numerous cryptographic implementations (see [MOP07, And08, KJJR11] and the references therein), both on small embedded devices and, more recently, on complex devices such as laptops, PCs, and smartphones [CZP, GPP+16, BFMRT, GPP+, CRR+, CMR+, GPT]. Physical emanations were used to recover information from peripheral input/output devices such as screens [EL85, Kuh, EGKP], printers [BDG+] and keyboards [AAa, BWY, VP, HS, BCV, HS15, ZZT09, CCLT].

Attackers seeking to exploit such channels face a challenge: attaining physical proximity to the target computer, in order to acquire physical measurements. In many settings, physical

*Authors are ordered alphabetically. Daniel Genkin, Roei Schuster and Eran Tromer are members of the Check Point Institute for Information Security.

access is controlled, and attempts to attain proximity will be blocked or detected. Alternatively, the attacker can seek to control suitable sensors that are already located in close proximity to the target; this may be tractable when there are ubiquitously deployed commodity devices, with suitable sensors, that can be adversarially controlled; for example, one of the low-bandwidth acoustic attacks [GST] can be conducted via a smartphone, using a malicious app that records audio using the built-in microphone when the smartphone is placed (for an hour) near the target.

We raise a third possibility: are there physical side-channel attacks for which the requisite physical measurements are readily available, and are shared by victims with untrusted parties as a matter of course, and yet the victims have no reason to suspect that they inadvertently leak private information?

We observe a new physical side channel that facilitates such an attack: *content-dependent acoustic leakage from LCD screens*. This leakage can be picked up by adjacent microphones, such as those embedded in webcams and some computer screens. Users commonly share audio recorded by these microphones, e.g., during Voice over IP and videoconference calls. Moreover, the pertinent sounds are so faint and high-pitched that they are well-nigh inaudible to the human ear, and thus (unlike with mechanical peripherals) users have no reason to suspect that these emanations exist and that information about their screen content is being conveyed to anyone who receives the audio stream, or even a retroactive recording. In fact, users often make an effort to place their webcam (and thus, microphone) in close proximity to the screen, in order to maintain eye contact during videoconference, thereby offering high quality measurements to would-be attackers.

Exploiting this channel raises many questions: What form of content-dependent acoustic signals are emitted by screens? Can these emanations be used to detect screen content? What measurement equipment and positioning suffices to measure them? Can they be acquired remotely via Voice over IP applications, despite the signal conditioning and lossy codecs employed by such applications?

1.1 Our results

We observe the existence of the aforementioned synesthetic side channel: "hearing" on-screen images. We characterize this content-dependent acoustic leakage on numerous LCD screens (with both CCFL and LED backlighting) of various models and manufacturers. The leakage has existed in screens manufactured and sold for at least the past 16 years, old and new models alike.

We show that this leakage can be captured by:

- Webcam microphones (see Figure 3.3).
- Mobile phones in proximity to the screen (Figures 3.2).
- "Smart speaker" virtual assistant devices (Figure 3.4).
- The built-in microphones of some screens.
- A parabolic microphone from a 10-meters line-of-sight to the back of the screen (see Figure 3.1).

Moreover, the leakage can be observed and analyzed:

- In archived audio recordings.
- From the remote side of a Google Hangouts video-conference call.
- In the cloud-stored audio recordings saved by virtual assistants.

We demonstrate exploitation of the above attack vectors for several attack goals:

1. Extracting text displayed on the screen (in a large font).
2. Distinguishing between websites displayed on the screen, or between websites and a videoconference screen.
3. Extracting text entered via Ubuntu's on-screen keyboard. (This shows that on-screen keyboards are not an acoustic-leakage-resistant alternative to mechanical keyboards.)

Our implementation of these attacks uses tailored signal processing combined with deep neural networks. We observe that the leakage characteristics, and the trained trained neural-network classifiers, often generalize across screens (even of different models). The attacks are robust to office-environment noise-level from nearby equipment such as computers, other monitors and human speech.

1.2 Related work

Physical side channels. Numerous prior works have demonstrated physical side channels, as discussed above. Categorized by channels, these include electromagnetic radiation [QS, GMO, EL85, Kuh, EGKP]; power consumption [KJJ, KJJR11, MOP07]; ground-potential fluctuations [GPT]; timing (often observable by any of the above) [Koc, BB05, BT]; and acoustic emanations from keyboards [AAa, BWY, VP, HS, BCV, HS15, ZZT09, CCLT], printers [BDG+] and CPU power supplies [GST]. Acoustic emanations are also mentioned in NACSIM 5000 "TEMPEST Fundamentals" [Nat] and related US government publications, but only in the context of electromechanical devices (as described in [Nat]: "[...] mechanical operations occur and sound is produced. Keyboards, printers, relays — these produce sound, and consequently can be sources of compromise"; see [GST] for further discussion).

While some physical side channels have been thoroughly explored, the only previous work extracting information from involuntary acoustic leakage of electronic components is Genkin et al.'s acoustic cryptanalysis [GST] work, which exploits coil whine from laptop computers, and does not consider acoustic leakage from displays.

Screen emanations. Extraction of screen content via electromagnetic emanations ("Van Eck phreaking" and screen "TEMPEST") is well known and studied, originally for CRT screens [EL85], and later also for modern LCD screens and digital interfaces [Kuh, Kuh03]. While the devastating potential of screen EM attacks has been explored in previous literature, acoustic emanations—whose traces are widely available and remotely attainable, and are thus exploitable by much weaker attackers—have not been addressed.

1.3 Outline

This paper is organized as follows: Section 2 characterizes the content-dependent acoustic leakage from various screens, and suggests a signal processing scheme for producing clean leakage traces. Section 3 introduces the attack vectors evaluated in this paper. We then discuss several attack scenarios, categorized by the attacker's goal: an on-screen keyboard snooping attack (Section 4), a text extraction attack (Section 5), and a website-distinguishing attack from various vantage points, including a remote attack over a Hangouts VoIP call (Section 6). Section 7 explores generalization of leakage characteristics, as modeled by the attacker, across screens and screen models. Section 8 discusses mitigations, and Section 9 concludes.

2 Characterizing the signal

In this section we explore the acoustic leakage signal emitted by various LCD computer screens (with both CCFL and LED backlighting) as well as attempt to characterize the connections between the leakage signal and the displayed image. We begin by providing some background about the process of rendering an image on modern LCD screens as well as by describing the experimental setup used in this section.

2.1 Background and experimental setup

Image rendering mechanism. Computer screens display a rectangular $l \times n$ matrix of pixels. Each pixel is typically further divided into red, green, and blue sub-pixels where the intensity of each sub-pixel is an integer between 0 and 255, thereby allowing the color of pixel to be uniquely represented using 24-bit integers. The screen's *refresh rate*, r, determines how many times per second an image to be displayed is sent to the screen by the computer's graphics card. The screen then renders the received image by iterating over the pixel rows from top to bottom, with the value of the pixels in each row rendered from left to right. Notice that the screen always re-renders the image it displays (using the above-described method) r times per second, even if no changes occurred in the image to be displayed. Typically, screens are refreshed approximately 30, 60, or 120 times per second, with a refresh rate of approximately 60 Hz being the most common.

Experimental setup. For experiments performed in this section we recorded the acoustic noise emanating from various monitors using a Brüel & Kjaer 4190 microphone. The microphone is connected to a Brüel & Kjaer 2669 preamplifier which ensures that the faint analog signals produced by the microphone correctly travel through the microphone's cable. We then amplify the analog signal to the level that it can be correctly digitized using a Brüel & Kjaer 2610 amplifier. Next, we digitize the signal using a Creative E-MU 0404 USB sound card sampling rate of 192 kHz, producing a .wav file. Finally, when needed, we visualize the spectrograms of the obtained signals using Baudline signal analysis software.

Vsync probe. For the purpose of signal exploration we shall sometimes utilize a trigger signal corresponding to the start of each screen refresh period. We implement this trigger by constructing a VGA tap cable that exposes the VGA's vsync line which (as specified in the VGA standard) carries a short 0.5 Volt pulse at the start of each screen refresh cycle. Note that the vsync probe is used for signal exploration purposes and is not necessary for the attacks presented in this paper.

2.2 Exploring the leakage signal

Distinguishing various images. We begin our analysis of acoustic leakage emitted from LCD computer monitors by attempting to distinguish simple repetitive images displayed on the target monitor. For this purpose, we created a simple program that displays patterns of alternating horizontal black and white stripes of equal thickness (in pixels), which we shall refer to as *Zebras*. The *period* of a Zebra is the distance, in pixels, between two adjacent black stripes. Finally, we recorded the sound emitted by a Soyo DYLM2086 screen while displaying different such Zebras. See Figure 2.1 for a picture of our recording setup.

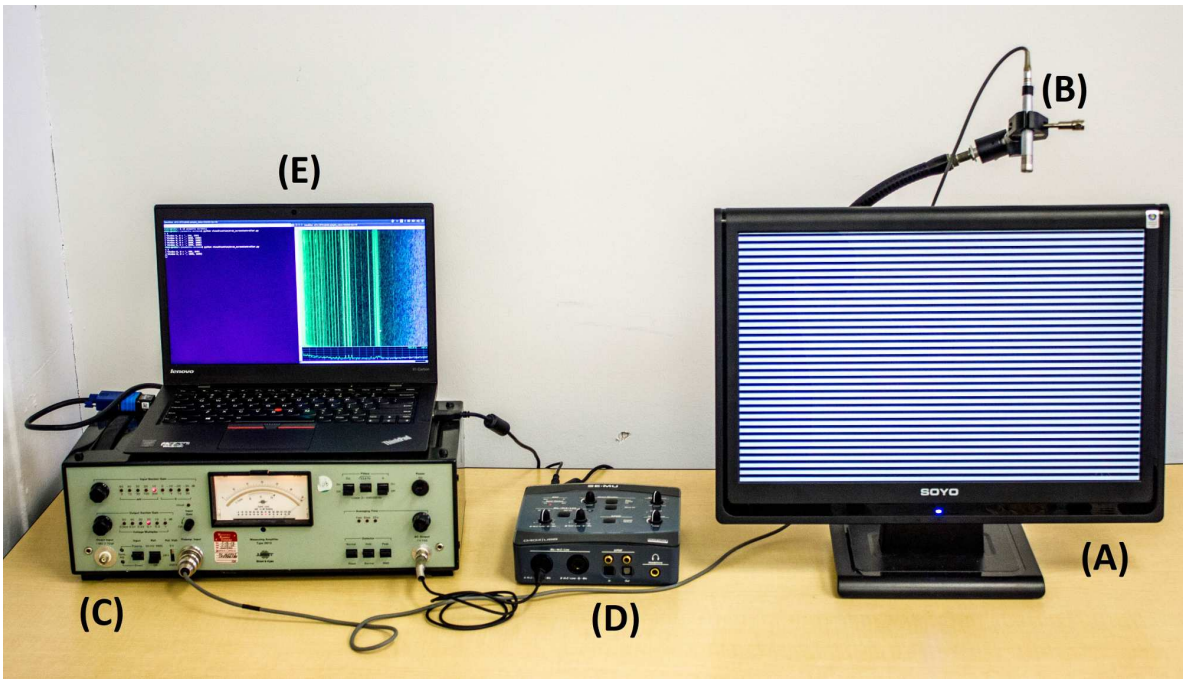


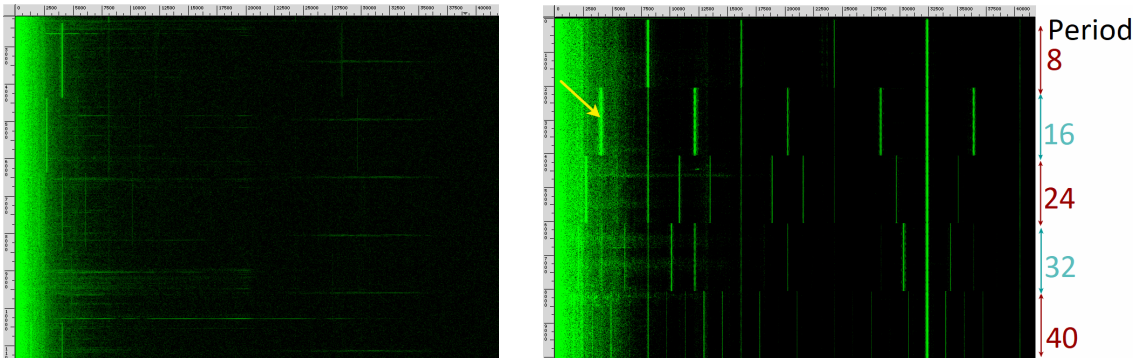
Figure 2.1: Setup used for signal characterization. In this photograph, (A) is a Soyo DYLM2086 target screen, (B) is a Brüel & Kjaer 4190 microphone, connected to a Brüel & Kjaer 2669 preamplifier, (C) is a Brüel & Kjaer 2610 amplifier, (D) is a Creative E-MU 0404 USB sound card with a 192 kHz sample rate (E) is a laptop performing the attack.

As can be seen from Figure 2.2–b, the change between the displayed Zebras causes clear changes in the monitor's acoustic signature. See Figure 2.3 for some additional examples of monitors that displayed particular clear signals. Beyond those presented in figures throughout the paper, we experimented with dozens of other monitors, including both old and newer-generation monitors of various sizes, and observed similar effects. We thus conclude that the leakage has existed in screens manufactured and sold for at least the past 16 years, old and new models alike, including LED and CCFL-based screens.

Acoustic or EM? In order to verify that the obtained signal indeed results from an acoustic signal emanating from LCD monitors, as opposed to electromagnetic radiation accidentally picked up by the microphone, we simply placed a sheet of non-conductive sound-absorbing material (e.g., thick cloth or foam) in front of the microphone. Figure 2.2–a is an analogy of Figure 2.2–b using the same experimental setup, but with the microphone physically blocked with a thick piece of cloth. As can be seen in the figure, the resulting signal is severely attenuated, thus proving that the signals observed here are indeed acoustic.[1]

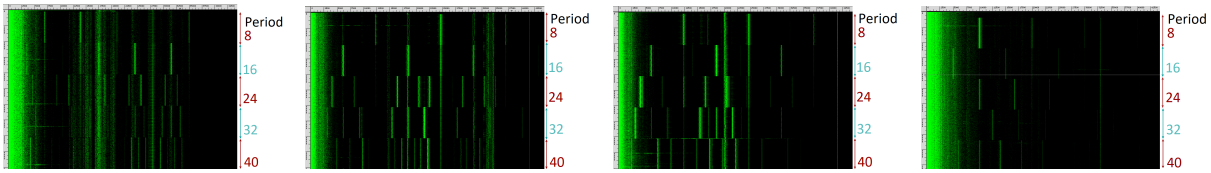
Physical leakage source. Having established the acoustic nature of the leakage signal, we have attempted to locate its source within the internal display electronics. To that end, we

[1] Conversely, with some of the microphones used in the self-measurements experiments of Section 3.3, the observed signal appears to have both acoustic and electromagnetic contributions (judging by the above methodology). This is, presumably, due to poor shielding and grounding on those cheap microphones. However, in the self-measurement setting, the precise physical channel through which the emanation propagates is inconsequential.



(a) covered microphone (b) uncovered microphone

Figure 2.2: (right) Spectogram of acoustic signals emitted by Soyo DYLM2086 screen while displaying Zebra patterns with different periods. Notice the difference in the screen's acoustic signature caused by the change in periods of the displayed Zebra pattern. (left) Spectogram of a recording taken under identical conditions, however, with the microphone covered by a thick piece of cloth. In both spectograms, the horizontal axis is frequency (0-43 kHz), the vertical axis is time (10 sec), and intensity is proportional to the instantaneous energy in that frequency band. The yellow arrow marks the 4 kHz leakage signal that is expected to be produced by a Zebra pattern with a period of 16 pixels.



(a) Samsung 920NW (b) HP ZR30w (c) Dell U3011t (d) Philips 170S4

Figure 2.3: Acoustic emanations of various target screens while displaying Zebra patterns of different periods, recorded using our setup from Section 2.1. The horizontal axis is frequency (0-43kHz) and vertical axis is time (10 seconds).

have disassembled a ViewSonic VA903b LCD monitor, which has a simple, modular internal design with a moderate number of internal components. As can be seen in Figure 2.4, in addition to the LCD panel itself (A) the ViewSonic monitor has two main boards: a digital board (B) which is responsible for implementing the monitors logic and picture rendering functions and a power supply board (C) which provides stable voltage to the digital circuits and backlight but does not directly process the screen's content.

In an attempt to locate the component producing the acoustic leakage, we measured the acoustic signals in various locations on both boards while displaying Zebra patterns on the LCD panel. By such acoustic probing, we localized the source of emanation to within the monitor's power supply board, in the area that is responsible for supplying power to the monitor's digital board (circled in green). We were unable to localize the source down to a single component, presumably due to acoustic reflections, diffraction and mechanical coupling

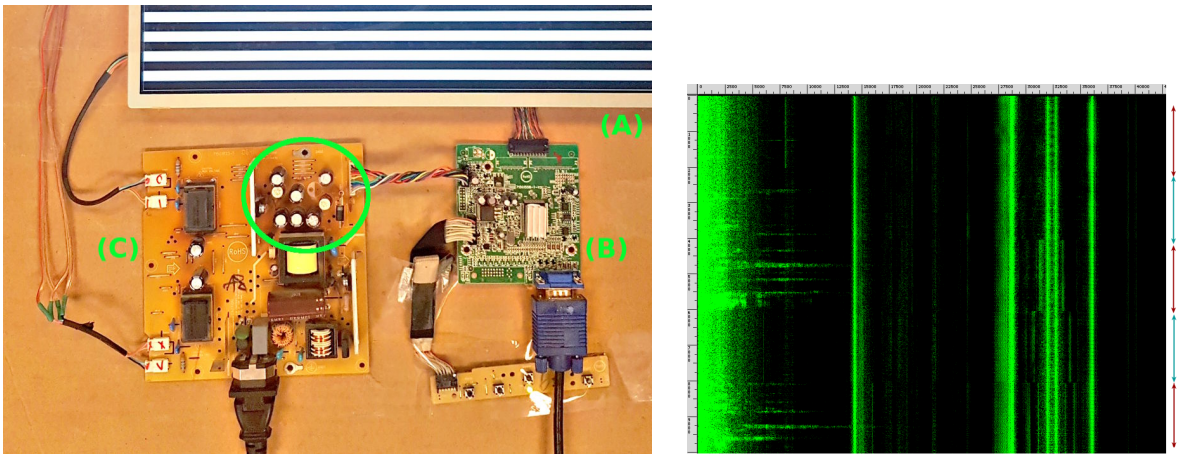


Figure 2.4: Internal components of a ViewSonic VA903b monitor. (A) is the LCD panel, (B) is the screen's digital logic and image rendering board and, (C) is the screen's power supply board. The green circle marks the approximate source of the acoustic signal. On the right is the acoustic Zebra leakage (similarly to Figure 2.3) from the disassembled screen.

in the board.[2]

Leakage Mechanism. We conjecture that the momentary power draw, induced by the monitor's digital circuits, varies as a function of the screen content being processed in raster order. This in turn affects the electrical load on the power supply components [KJJR11] that provide power to the monitor's digital board [EGKP], causing them (as in [GST]) to vibrate and emit sound. Unfortunately, the precise propagation of signal along this causal chain is difficult to precisely characterize: power signal modulation have a complex dependence on circuit topology, layout and environment; inadvertent power-to-acoustic transduction varies greatly with circuits electrical and mechanical characteristics; the different channels involve different means of acquisition and have different bandwidth and SNR constraints, necessitating different signal processing; and the choice of acoustic sensors and their placement creates additional considerations that cannot be separated from the transduction's physical nature. We thus focus on the aggregate bottom-line effect and its exploitability in the practical scenarios (e.g., webcams and phones).

Analyzing the leakage frequency. Having verified that different images produce different acoustic leakage, we now turn our attention to the connection between the displayed image and the produced leakage signal. Indeed, assume that the rendering of individual pixels produce sounds that depend on the individual pixel's color (e.g., black or white). Because the entire screen is redrawn approximately $r = 60$ times per second, we expect that spatial-periodic changes in the displayed pixels will introduce a strong frequency component corresponding to the frequency of transitions between rendering black and white pixels.

More specifically, a Zebra with a 16-pixel period drawn on a screen with a vertical resolution of 1050 pixels produces $1050/16 = 65.625$ color changes per screen redraw. Accounting for approximately 60 redraws per second, the acoustic leakage is expected to have a strong $131.25\cdot$

[2] In an attempt to mitigate these, we also tried to lift major components away from the board and connect them by extension wires; but this changed the circuits behavior and resulting signal, and did not yield conclusive results.

$60 = 3937.5$ Hz frequency component; which is clearly observable in Figure 2.2–b (marked by a yellow arrow).

2.3 Signal analysis in the time domain

Following our hypothesis that the intensity of pixel lines is directly manifested in the leakage signal, we monitored the leakage of a Soyo DYLM2086 screen (see Figure 2.5) while displaying a Zebra image with a period of 21 pixels. However, this time the Zebra was modified in two ways. First, the intensity change between stripes is smoothed, following a sinusoidal pattern of the given period. Second, we replaced the middle third of the picture with a solid black area (see Figure 2.5–a).

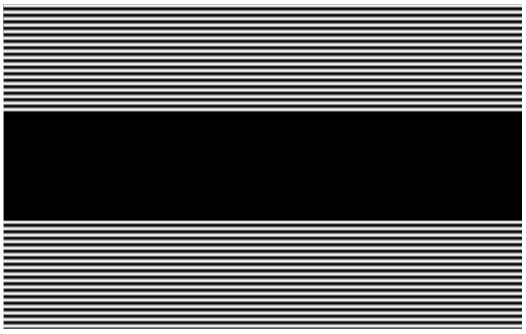
As can be seen in Figure 2.5–b, displaying such a modified Zebra causes a strong leakage at the Zebra's "natural" frequency of $1050/21 \cdot 60 = 3000$Hz on the resulting spectrogram. Next, plotting the acoustic signal in the time domain (after applying a band-pass filter at 3-4kHz) has resulted in a 3 kHz signal (matching the Zebra's frequency), where the amplitude of the middle third of the signal is considerably lower compared to the first and last third (see Figure 2.5–c). Experimenting with different sizes for the solid black area, the lower-amplitude area increases and decreases correspondingly. We conclude that the momentary amplitude of the acoustic leakage signal is a rather accurate approximation of the brightness level of the individual pixel rows of the target monitor.

2.4 Analyzing amplitude modulated signals

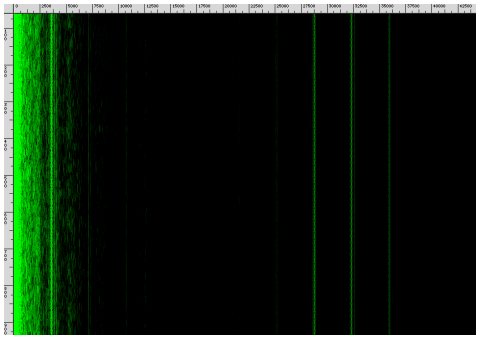
Observing the spectrogram in Figure 2.2–b, we notice that in addition to signal changes in the $0 - 22$ kHz range which correspond to different Zebra patters, there are also changes at the $27 - 37$ kHz range which are again correlated to different Zebra patters. Analyzing the latter range, we notice that the signal takes form of two distinct side lobes that mirror a central 32 kHz carrier and that increasing the period of the Zebra image being displayed leads to the carrier's side lobes coming closer together. This signal behavior at the $27 - 37$ kHz range is highly indicative of a different type of leakage (compared to the $0 - 22$ kHz range) where an amplitude modulated signal being transmitted on a central carrier. Investigating this hypothesis, in Figure 2.5–d we plot in the time domain the signal obtained after applying a 27.5-38 kHz bandpass filter around the carrier and its two side lobes. As can be seen, there is a clear correlation between the color of the displayed pixel row and the amplitude of the 32 kHz carrier (blue), where dark rows corresponding to high signal amplitudes. Recovering the envelope signal (red line in Figure 2.5–d) using the Hilbert transform method [Hah96] results in a direct leakage of the color of the displayed pixel row.

Leveraging the modulated signal. As the modulated signal is present on a carrier frequency that is much higher than the frequency of most naturally occurring acoustic signals (e.g., human speech), it can be leveraged to extract a relatively clean version of the screen's image. We now proceed to describe our method for performing acoustic image extraction.

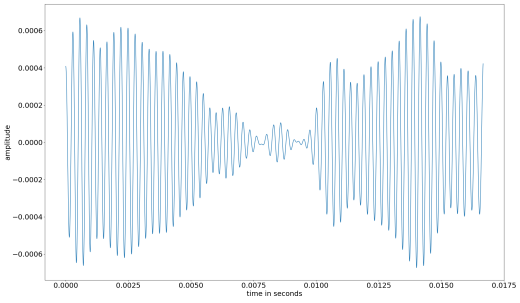
Because the screen refreshes approximately 60 times a second, the *output trace* produced by our extraction process is a 192 kHz time series vector in the duration of one refresh cycle (1/60 s). Ostensibly, we simply need to sample the demodulated signal for one refresh cycle. However, as the leakage signal is not completely noise free, we leverage the fact that the entire image is redrawn 60 times per second (thus producing a fresh sample of the leakage signal



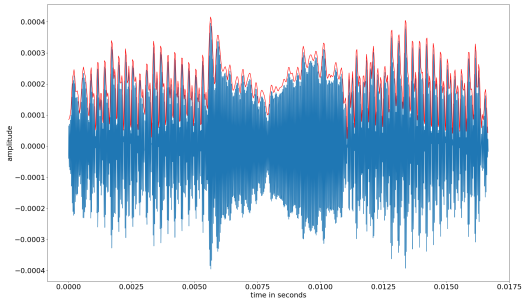
(a) A Zebra image with a period of 21 pixels and a sinusoidal color transition with its middle part replaced by a solid black area.



(b) Spectogram (0-43 kHz, 800 ms) of the resulting acoustic signal. Notice the strong acoustic signal present at 3 kHz.



(c) A segment of the acoustic signal after applying a 3-4 kHz bandpass filter. Notice the clear correlation between the amplitidue of the red signal and the brightness of pixel lines.



(d) Blue: a segment of the acoustic signal after applying a 27.5-38 kHz bandpass filter around the screen's carrier signal. Red: amplitude demodulation of the blue signal. Notice the inverse correlation between the amplitude of the red signal and the brightness of pixel lines.

Figure 2.5: Analyzing the acoustic leakage of a Soyo DYLM2086 screen. Recording captured using our setup from Section 2.1. Figures 2.5–c and 2.5–d are time-synchronized using the vsync probe.

corresponding to the image at each redraw) in order to reduce noise by averaging. More specifically, after recording the screen's acoustic emanations for a few seconds, we bandpass filter the obtained recording around the carrier signal (allowing only the frequencies 27.5–38) and AM-demodulate it by subtracting the trace's average, computing the analytic signal using a Hilbert transform [Hah96], and taking its absolute value. We then divide the signal into *chunks* of samples where each such chunk contains the acoustic leakage produced during a single screen refresh cycle. Finally, we perform sample-wise averaging of the chunks.

2.4.1 Denoising challenges.

Our method of choice is chunking and averaging. In order for noise to cancel out during averaging, we need chunks to be very close to one another in length and phase (of the refresh period). One approach is to use the known screen refresh rate and segment the signal into 1/60 s chunks, each chunk corresponding to a refresh cycle. This presents two challenges:

Challenge 1: drift. Unfortunately, using exactly 60 Hz is usually erroneous by up to 0.2 Hz. The refresh rate used has to be extremely accurate: in our sample rate (192 kHz) and around a 60 Hz refresh rate, an error of about 0.002 Hz would cause a drift of one sample per ten chunks. Even such a small error would introduce destructive interference in averaging hundreds of chunks. The attacker thus has to very accurately measure the refresh rate. Empirically, even for the exact same configuration, the refresh rate has slight changes of up to 0.02 kHz.

Challenge 2: jitter. Using the vsync probe (see Section 2.1), we examined refresh cycles. We observed signal jitter that caused destructive interference even when averaging with the accurate refresh rate. The jitter creates timing abnormalities that are not averaged out over the span of a few seconds. For our Soyo monitor, every few dozen refreshes, there is an *abnormal cycle*, whose duration is abnormal. The abnormalities are erratic: cycles can be either down to 10% shorter than 1/60 s long, or as long as 1/30 s. Even with the accurate average refresh rate, there will still be drift: the average refresh rate is actually far from the refresh rate for non-abnormal refresh cycles. Effectively, every few hundred cycles, the refresh period gets phase-shifted by some unknown value.

Naive solution: correlation. A natural approach is first approximating or measuring the refresh rate, segmenting the trace into chunks corresponding to refresh cycle duration, and use correlation with one arbitrarily chosen "master" chunk to phase-align the traces, i.e., trace phases are aligned such that correlation is the highest. Then, outlier rejection can be performed to mitigate jitter and drift in still-misaligned chunks. We found the variants of this approach to underperform even when the refresh rate used is extremely accurate (see Figure 2.6a). Presumably, this is because they all allow for very lenient alignment of the chunks to the master chunk, i.e., all chunks are rotated so that they correlate best with the master chunk, which contains a noisy version of the signal of interest. Averaging is thus likely to increase the noise in the master chunk. Another problem is arbitrarily setting the master chunk, which can produce an abnormally noisy chunk. The approach is also prohibitively slow.

2.4.2 Our denoising approach.

We need a chunking algorithm that is robust to jitter and small drift, i.e., every chunk should correspond to a refresh cycle at a specific phase (identical for all chunks). Denote the sample rate f_s. We empirically observe that, except for abnormal cycles, refresh times are relatively stable in the following sense: for each of our recordings there exists an integer $W \approx \frac{f_s}{r}$ such that cycles are either W or $W+1$ samples long, implying that the actual refresh cycle is between $\frac{f_s}{W}$ and $\frac{f_s}{W+1}$. Moreover, W is always one of two possible consecutive numbers: $S, S-1$. Non-abnormal cycle sizes are therefore always in $S-1, S, S+1$. Another empirical observation is that Pearson correlation values can be used to heuristically distinguish between pairs of chunks with the same cycle phase and ones with different phases: same-phase chunks are typically above some threshold value, whereas different-phase chunks are typically below it (especially if the phase difference is more than a few samples). Our chunking algorithm is parametrized by S, by a small integer d, and a "correlation threshold", a real number T.

High-level overview. The algorithm starts from the first sample in the signal, and iteratively finds the next chunk start location using Pearson correlation with a reference chunk, assuming that each chunk size is in $G = \{S-d, ..., S+d\}$. The range d is very small: in our experiments it is usually 1, and never more than 3. This is designed to make small adaptive changes in chunk sizes to account for the minor drift introduced by the refresh-rate approximation

and, sometimes, misalignment made in the previous round. When correlation of the reference chunk with the next chunk drops below T, we enter a "sync" mode where the purpose is to re-synchronize the chunk phase, again using correlation. If the algorithm enters sync mode for more than 15% of the signal chunks, or if the algorithm does not exit sync mode after three iterations, it quits with an error. After averaging, we rotationally shift the resultant array such that the highest value is the first.[3] Appendix A.1 describes our algorithm in detail.

Choosing parameter values. d is chosen to be very small to ensure we are not increasing the noise by maximizing correlation (see advantages below). We used 3 for the keyboard snooping and cross-screen attacks, and 1 for the other ones. T should be the threshold that differentiates best between out-of-phase chunk pairs and in-phase chunk pairs. For clean signals such as the close-range attack, we chose 0.9. For noisier signals we chose 0.4-0.8, accepting some "false" entries into sync mode. We measured $S = 3206$ to $S = 3202$ empirically using the vsync probe. The attacker, however, does not need to attach a probe to the victim computer: since the range of possible values is small, the attacker can apply brute force guessing and, using our algorithm, choose the value of S that maximizes average chunk correlation with the master chunk and minimizes sync mode iterations. This approach accurately finds the correct S value using under a minute of victim screen recordings. In Section 7 we simulate a cross-screen attack without attaching a vsync probe to the victim screen(s).

Advantages of this approach. First, it is less likely to augment the noise introduced by the master chunk. This is because of its strict limitation: the vast majority of chunks found have to be consecutive and with a very tight size constraint. Only the first chunk in each sequence is leniently aligned such that it correlates best with the master chunk.

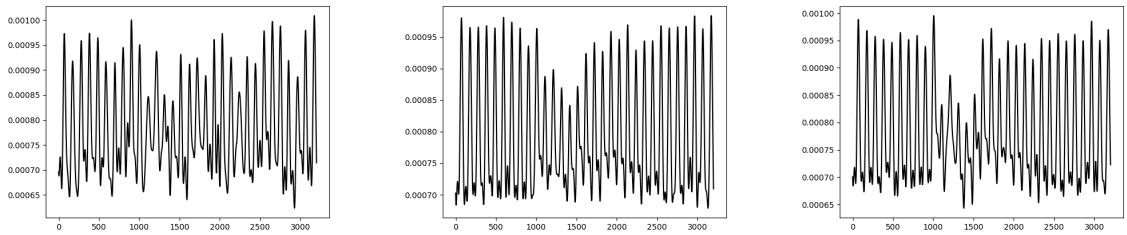
Second, the master chunk is not arbitrarily found, but is a chunk that correlates well with at least its consecutive chunk. An abnormally noisy chunk is less likely to be selected.

Third, the algorithm is not parametrized by the exact refresh rate. Instead, it can use an approximation. Not depending on the exact refresh rate is crucial, since it slightly changes with time, so the attacker cannot be expected to track it.

Finally, this algorithm is much faster than the rotational-shift-based baseline. The complexity of both methods is dominated by the required Pearson correlation computations, an operation with complexity $O(S)$. The baseline's approach computes the Pearson correlation and performs this $C \times S$ times, where C is the number of chunks. For a 5 s trace sampled at 192 kHz, our implementation of this approach takes 166 s on an Intel Xeon E5-2609 v4. While in the worst case ($6000 \times 0.15 \times C$ correlation operations in sync mode, and another $2 \times d \times 0.85 \times C$ correlation operations in normal mode), our algorithm performs only about three times fewer correlation computations as the baseline; in practice it is over two orders of magnitude faster on average. Even for recordings captured using a parabolic mic, where the signal is noisy and sync mode is entered relatively often, the average processing time is less than 2 s for 5 s recordings.

Figure 2.7 shows the output trace for the black hole image. Figure 2.6 compares our method with the baseline, naive one, for processing a particularly noisy Zebra signal recorded using a parabolic mic directed at a Soyo screen. The naive approach comprises of (1) chunking according to the the exact de-facto refresh rate of 59.9019, (2) rotationally shifting chunks according to maximal correlation, and (3) performing outlier rejection, removing chunks whose

[3]We found that the sample with the highest value corresponds roughly to one of a few specific refresh period phases. This results in relatively consistent trace phase alignment.



(a) Output of the naive approach.

(b) Output of our approach, using $S = 2306, T = 0.05, d = 3$.

(c) Output of "ground truth" chunking, using the trigger signal obtained via the vsync probe to accurately segment the trace.

Figure 2.6: Comparing the outputs of the two chunking approaches on a Zebra of 30 periods, recorded using a parabolic mic (see Section 3.1). The naive approach produces a signal whose amplitude is erratic and bears less resemblance to the one of the ground truth, even though it is using as input the exact refresh rate (up to 0.0001 s), and even though this rate is entirely stable across this particular recording (there are no abnormally sized cycles).

correlation is less than 0.05.

3 Attack vectors

We consider four possible attack vectors. For each vector, we specify our experimental setup, demonstrate that leakage exists and, in the following sections, evaluate screen content detection attacks.

3.1 Close-range and at-distance attacks

In this setting, the attacker uses a high-end microphone in order to extract the image displayed on the screen. While a relatively strong attack model compared to other models considered in this paper, it is most effective for estimating the extent of acoustic information leakage from various attack ranges.

Experimental setup. We used a setup similar to the one in Section 2.1. We added 21 kHz high-pass filter (Krohn-Hite 3940). For the distance attack, we mounted the microphone in a parabolic dish, placed about 10 meters away from the screen. See Figure 3.1 for an example of such a setup. To simulate a realistic scenario, recordings were taken in an office environment, with some noise from nearby equipment and people occasionally talking in the proximity of the microphone. We do not expect speech to interfere, as most of the leakage frequencies are well-above speech frequencies.

3.2 Phone attack

We consider a commodity phone placed in proximity to a screen, and recording the screen's acoustic emanations using the phone's built-in microphone (see Figure 3.2). This vector opens a host of attack scenarios such as an attacker using his personal phone in order to spy on the

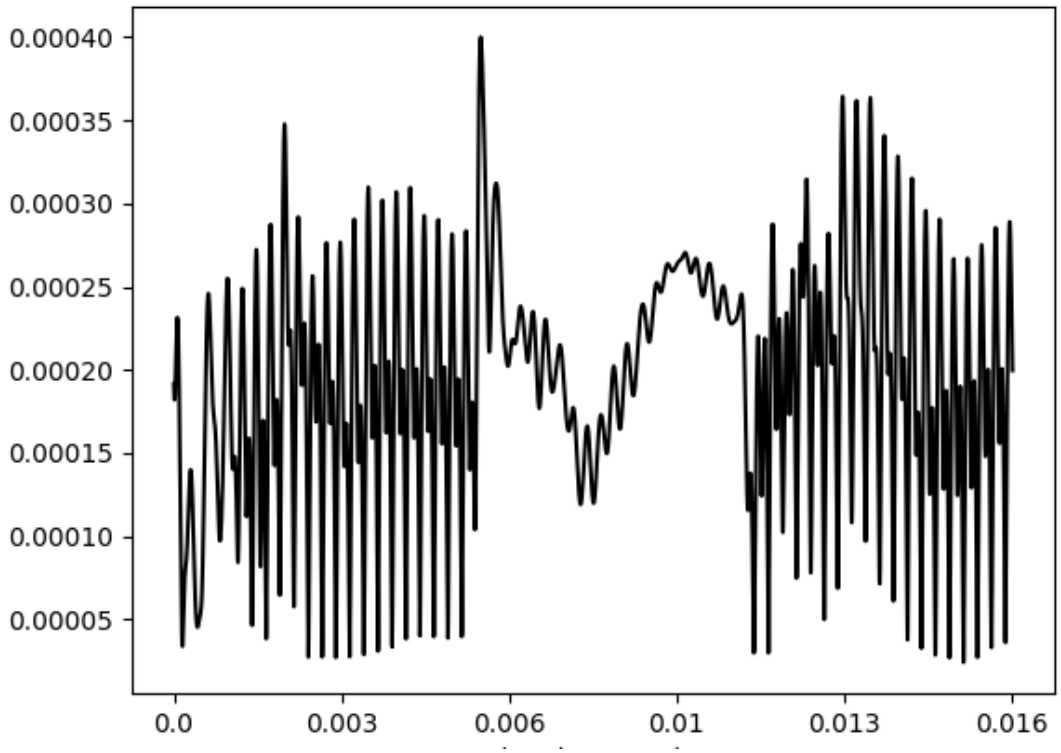


Figure 2.7: The output trace of the image in 2.5–a, using $T = 3206, S = 3206, d = 3$

content of someone else's screen that is turned away from him (e.g., in a business meeting). The attack can also be conducted remotely by a rogue app, without the phone owner's knowledge, because many mobile apps have permission to record audio at any time [ST, FHE$^+$].

Experimental setup. For this attack setup, we used an LG V20 phone directed at a screen and recording using its built-in microphone. We used the Amazing MP3 Recorder app [Ste] (configured to record at 192 kHz), which supports high- kHz configurations and and, uniquely, exports an interface we used for automating trace collection en masse: a "start recording" and "stop recording" are exported by this app and can be used by *any* other app on the device through Android's inter-app communication mechanisms. [4] We could thus use the Android Debug Bridge (ADB) interface for invoking recordings. Here, too, recordings were taken in an office environment with some environmental noise and human speech.

Demonstrating leakage. We found that the phone, recording at 192 kHz, can capture the leakage extremely well. We performed an experiment similar to those in Section 2.2 twice: once with the phone directed at the back of the screen (simulating a physical attack in, e.g., a business meeting), and once with the phone naturally positioned on a table near the screen (simulating a remote attacker, e.g., an app). In both cases, the measured screen was a Soyo DYLM2086 screen. Figure 3.2 shows the resulting spectrograms, containing the expected leakage pattern. While for the naturally positioned phone the signal is attenuated, it is still clearly visible on the spectrogram. The leakage signal for the directed position is dominant and is almost comparable to the leakage samples recorded using high-end equipment.

[4] The app exposes interfaces ("broadcase receivers") that can be used by any app, regardless of its permissions, to record audio. This demonstrates the commonplace nature of mobile based audio capturing adversaries.

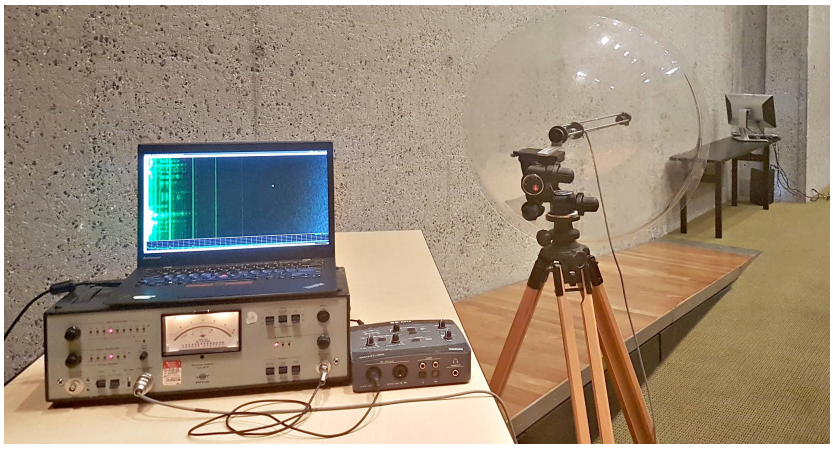
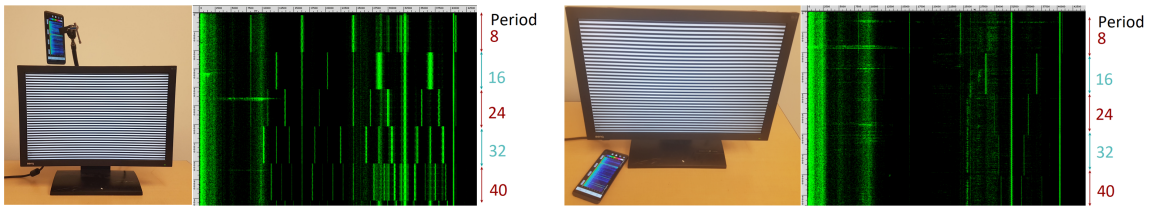


Figure 3.1: Extracting an image from a screen using a parabolic microphone at a distance of about 5 meters.



(a) Phone microphone directed at the back of the screen.

(b) Phone placed on a desk by the screen.

Figure 3.2: Acoustic emanations (0-43 kHz, 10sec) of a BenQ q20ws screen while displaying Zebra patterns of different periods, recorded using an LG V20 smartphone.

Evaluation. We evaluate attacks using this vector: on-screen keyboard snooping is evaluated in Section 4 and a website distinguishing attack is evaluated in Section 6.

3.3 VoIP attacker

We also consider a remote adversary who is conversing with the target over the Internet, using a Voice over IP (VoIP) or videoconferencing service. As long as the webcam (if any) is not pointing at the screen (or at its reflection of some object), the target would assume that remote participants in the call cannot glean information about the local screen's content. However, the adversary receives an audio feed recorded with the target's microphone, which is usually located in close proximity to the screen, or even embedded into the screen assembly itself (in order to maintain eye contact during video calls). This microphone picks up the screen's acoustic emanations, the VoIP service relays it digitally, and the remote attacker can (as we will show) analyze this signal in order to extract the screen's content.

Experimental setup. Empirically demonstrating this, we obtained the screen's acoustic emanations by recording the audio sent to a remote party during a Hangouts call, captured using the victim's own microphone (built into a commodity webcam). Here, too, recordings were taken in an office environment with some environmental noise and human speech.

More specifically, to simulate the victim, we used a PC running Ubuntu 16.04 connected to a Dell 22" LCD monitor (model 2208 WFPt) screen, and a Microsoft LifeCam Studio webcam. The camera was placed naturally by the screen, similarly to Figure 3.3–a. To simulate the attacker, we used a second PC running Ubuntu 16.04. We set up a Hangouts connection between the attacker and victim, both running Hangouts over a Firefox browser. At the attacker end, we redirected all sound output from the soundcard into a loopback device. We could then use `arecord` on the device to capture sound originating at the victim end of the call.

Observing the leakage. To check for leakage in this setting, we again performed measurements by acquiring traces while displaying alternating Zebra patterns, similar to those in Section 2.2, but using various webcams and screens. Figure 3.3 summarizes our findings.

We discovered that first, commodity webcams and microphones can capture leakage. Second, natural and expected positioning of cameras can be sufficient. In fact, even the built-in microphones in some screens (e.g., Apple LED Cinema 27-inch, see Figure 3.3–b) can be sufficient. Third, the leakage is present (and prominent) even when the audio is transmitted through a Hangouts call; see Figure 3.3–a.

Attack evaluation. See Section 6.2.

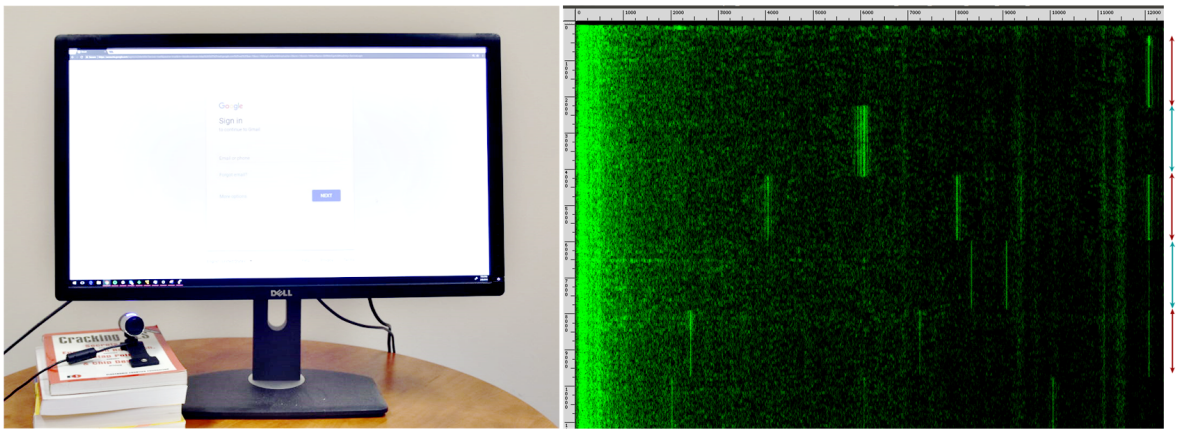
3.4 Virtual assistant / "smart speaker" attack

The contents of the user's screen can be gleaned by voice-operated virtual assistants and "smart speakers", such as the Amazon's Alexa assistant running on Amazon Echo devices and the Google Assistant running on Google Home devices. Once configured, such devices capture audio at all times, includign acoustic leakage from nearby screens. When a *wake phrase*, such as "Alexa" or "Hey Google", is detected by the device's elaborate microphone array, it enters an attention mode to recognize and response to commands. The recording of the wake phrase, as well as subsequent recorded audio until interaction stopped, is archived in cloud servers ("to learn your voice and how you speak to improve the accuracy of the results provided to you and to improve our services", according to Amazon [Tod18]).

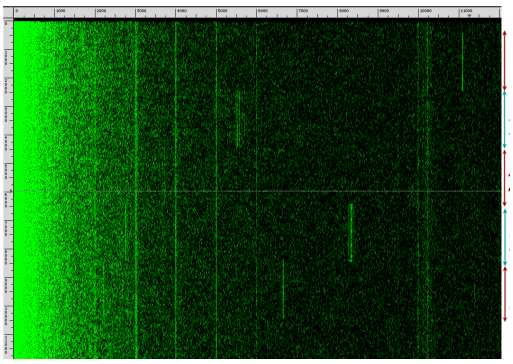
We set out to ascertain that the screen's visual content is indeed acoustically captured and subsequently uploaded to Google's and Amazon's cloud servers. First, we placed a Google Home device next to a Dell 2208WFPt screen displaying alternating Zebra patterns. We then woke the Google Home device up by using its wake phrase ("Hey Google") and kept the the recording running during the Zebra alternations. Finally, we retrieved the recorded audio from Google's servers. Figure 3.4 shows the spectrogram representation of the retrieved audio, where the alternation of Zebra patterns is clearly visible. We proceeded to perform the exact same procedure with an Amazon Echo (2nd Gen.), and observed similar results.

4 On-screen keyboard snooping

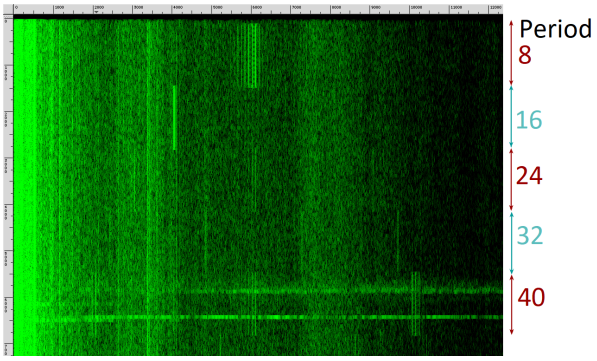
In this attack, the attacker aims to extract words or sentences from an on-screen keyboard. We experiment with horizontal and portrait layouts. We also assume the on-screen keyboard is sufficiently large, depending on screen attributes. On-screen keyboards (operated by touch or mouse) are offered by all mainstream operating systems, and also used by some websites



(a) Microsoft LifeCam Studio webcam positioned below a LED-backlit Dell U2713H screen, recorded through a Hangouts call. Note that the frequency axis range is 0-13 kHz, due to the lower sampling rate used by Hangouts.



(b) Self-measurement: an embedded microphone in an LED-backlit Apple LED Cinema 27-inch screen.



(c) Logitech C910 webcam positioned on top of a 20“ Soyo screen.

Figure 3.3: Time-frequency spectrum of alternating Zebra frequencies measured through various naturally positioned commodity devices. The spectrograms of all combinations indicate similar acoustic leakage patterns.

as a security mechanism for password entry, reducing the threat of keyloggers.[5] Using an on-screen keyboard also protects against attackers with acoustic probes that try to characterize individual key acoustic emanations [AAb, LWK+, ZZT09]. We show, however, that the pressed on-screen keys can be inferred from the screen's acoustic emanations.

4.1 Machine learning attack methodology

Our attack works in two stages: first, in an *off-line* stage, the attacker collects training data (audio traces) to characterize the acoustic emanations of a given type of screen, and uses machine-learning to train a model that distinguishes the screen content of interest (e.g., websites, text, or keystrokes). In the *on-line* stage, the attacker records an audio trace of the actual screen under attack (whether in person or remotely), and then uses the trained model

[5] On-screen keyboards resist keyboard logging by malware (unless the malware is adapted to log screen content), and also low-level keyboard device snooping, e.g., by leaky USB hubs [SGRY].

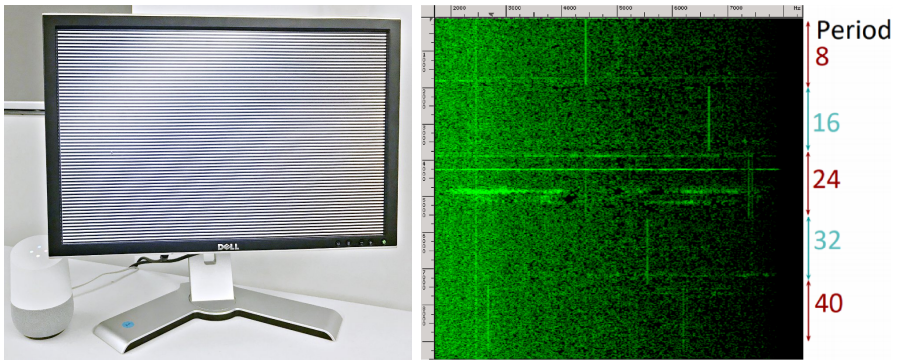


Figure 3.4: (Left) alternating Zebra patterns displayed on a Dell 2208WFPt monitor, recorded by a Google Home device and archived on Google's cloud. (Right) the spectrogram of the recorded signal. Note the frequency axis ends at 8 kHz, since Google's archived recordings play at 16 kHz sample rate.

to deduce the on-screen content.

Limitations. Machine learning models are sensitive to changes in the attack data with respect to the training data. For this reason, the simulated attack used for acquiring training traces should be similar to the actual attack scenario. Specifically, similar audio recording equipment and configuration should be used. Moreover, different monitor models display different content-dependent sound patterns, so the attacker must have precise knowledge of the user's monitor model, and be able to procure it.

Why machine learning? The algorithm in Section 2.4 produces relatively clean *output traces*, where sample values are clearly dependent on the screen's content.

We know that the values are particularly sensitive to pixel line intensity, and that they are higher as intensity increases. We could use this to try extracting pixel line intensity values, and use those directly as website fingerprints, and to extract text, etc. However, the correspondence between pixel line intensity and sample values is intricate and difficult to accurately model. For example, sample values seem to depend not only on their corresponding pixel line intensity, but also on previous sample values. Accurately modeling this dependence would most likely depend heavily on the attributes of the specific monitor used. Moreover, it is unclear that pixel line intensities are the only thing that leaks onto this signal. If there is any horizontal pixel information, we would like to leverage it, too. Finally, although we produce relatively clean output traces, they do contain noise. Two output traces of recordings of the same screen content will not be entirely the same.

Neural networks, and specifically convolutional neural networks (CNNs) [Wik], are very good at supervised learning on time series data, even if it is noisy [ST]. In many cases, when CNNs are configured correctly, we can expect them to be able to directly perform the processing task at hand, at least as long as it is a supervised learning task and enough data is available.

Using classifiers. We employ a CNN-based architecture for solving our task. We define, train, and use CNN *classifiers*: a classifier's input is a sample, represented (in our case) as a time-series vector. Specifically, our classifier inputs will be the outputs of the signal processing procedure in Section 2.4. A classifier's output is a vector of probabilities, one for each class. A sample's *prediction* is the class with the highest assigned probability. Before the classifier can

produce meaningful output, it has to be trained. The training procedure feeds the classifier with sample traces and correct predictions.

Simpler ML architectures. We fine-tune our neural networks for speed and accuracy. Alternatively, our traces (after signal processing) are often clear enough for classification by a rudimentary logistic regression model (at the cost of a much slower training, due to slow convergence). For the lower-frequency non-modulated signals acquired through Voice over IP and used without trace averaging (Section 6.2), the rudimentary model does not converge, but a carefully tuned CNN is effective.

4.2 Attack simulation and results

We simulated this attack in the smartphone and close-range attack vectors.

Data collection and preprocessing. We captured audio recordings of a Soyo 20" DYLM2086 in the close-range and smart phone settings (Section 3.1 and 3.2, while displaying screen shots of the mouse pressing varying keys. We used the native Ubuntu on-screen keyboard with a phone-style layout, "High Contrast" theme and size 1050x825. We used portrait keyboard layout (see discussion below).

For each attack vector, we iterated 100 times over the 26 letters of the alphabet. In order to reliably simulate the acoustic leakage during hundreds of key-presses on the on-screen keyboard required for training, we used screenshots containing the on-screen keyboard with the respective key being pressed. In each iteration, for each key, we collected a 0.5 s long trace when key is pressed (simulated by the screen shot). We split these traces to train (90%) and validation (10%) sets. [6]

For testing our snooping attack, we also recorded audio while words are typed on the virtual keyboard. Words were chosen randomly from from a list of the most common 1000 English words [Nor] (see results). We use the traces of 50 such words for testing both classifiers. Characters of the words were typed consecutively, with 3 s between each transition. We also experiment with words recorded at 1 s speed (see results).

We applied our signal processing procedure in Section 2.4.

Training procedure. Trace processing and training was performed on an Intel(R) Xeon(R) CPU E5-2609 v4 processor with two Titan X GPUs. We uses the Keras Python library with a Tensorflow backend to program and train the networks. Keras is used with a Tensorflow 1.4 backend and CUDA 8.0, CUDNN 6.0.

Each on-keyboard keystroke results in a specific image displayed on the screen. We can train a classifier to identify the different characters for a given trace. However, some pairs of keys are completely horizontally aligned on this virtual keyboard. We do not expect to be able to differentiate between them. We group each of those pairs to one class label. Our labels were thus 'b', 'c', 'm', 'n', 'p', 'v', 'x', 'z', 'aq', 'sw', 'de', 'fr', 'gt', 'hy', 'ju', 'ki', 'lo', and space.

CNN architecture: We used a convolutional layer with window size 12, 16 filters, and stride 1, followed by another convolutional layer with the same parameters but 32 filters, followed by a max-pooling layer with pool size 4, and one fully-connected layer with output size 512. The convolutional and FC layers have ReLU activations. The last layer is a softmax. **Hyperparams:** we used an Adadelta optimizer with a categorical crossentropy loss, and a

[6] Here and in Section 5, we use just 10% of the initial recordings for validation. In both, validation sets are still in the hundreds, and we record an additional trace set, 100% of which was used for testing.

batch size of 64. We trained each network for 200 epochs. Each epoch takes around 3 s. The model was evaluated on the validation set after every epoch; the best-performing model was saved for testing.

Testing procedure. To process a trace, we shift a 0.5 s window across the duration of the trace. The window offset is advanced in 3200 sample (1/60 s) intervals. For each offset, we apply our processing algorithm on the sub-trace contained in the window. We then apply our classifier.

The output of this process is an array of class labels of size $60 \cdot l - 30$, where l is the original trace length. We traverse it from start to finish and output class labels appearing more than 15 times contiguously for the smart phone attack and 35 times contiguously for the close-range attack. Assuming the CNN always predicts the right class, this produces sequences of class labels that correspond to the letters typed on the screen. However, these sequences do not distinguish (1) letters grouped into the same class, and (2) letters in the same class that were typed sequentially. For example, for a trace of the word "love", we would expect that the sequence ['lo', 'v', 'de'] be produced. The 'l' and 'o' keys are grouped into the same class, and there is only one instance of this class label. Similarly, the expected trace of the word "screen" is ['sw', 'c', 'fr', 'de', 'n'].

To disambiguate each produced trace, we go over the 102000 words in the dictionary. For each, we compute its expected trace, and check it against the trace. We return the list of words that matched the trace, or the *prediction list*.

Results. Our classifier reaches 100% accuracy on the validation set, for both the smartphone and close-range. For the close range attack's word traces, the correct word was always contained in our prediction list. The size of the prediction list varied from 2 to 23 with an average of 9.8 words per trace. For the smartphone attack, the correct word was contained in the prediction list for 49 out of 50 traces.

To further test the limits of this attack, we repeated the attack using a different screen, Soyo 22" (model DYLM2248), in the close-range setting. Again, the validation set accuracy was 100%. This time, we collected traces of 100 additional words, twice: once while waiting 3 seconds between each transition, and once when waiting 1 second. For the 3 s collection, the correct word was in the prediction list 94 times out of 100. For the 1 s recordings, the correct word was in the prediction list 90 times. The average candidate list size was 8.

Portrait vs. landscape layout. For leakage of pixel line aggregate intensities (see Section 2.3), keys that are horizontally aligned (in raster scan order) contribute to the same line intensities and are thus indistinguishable. It would thus be easier for an attacker to infer keys when the keyboard is in portrait mode: in each row of keys, every key uses different raster lines, so there is no ambiguity within a key row. Indeed, the experiments above used portrait mode. However, we also evaluated the attack in landscape (normal) orientation, where intra-row ambiguity occurs. We repeated the collection of data using the Soyo 22" in landscape keyboard layout. We trained the classifier on both this landscape and the aforementioned portrait datasets, but without grouping indistinguishable letters (which would leave the landscape set with just 3 classes, one for each key row). The portrait layout classifier reaches 96.4%, and the landscape layout classifier reaches 40.8%. Top-3 accuracies are much higher: 99.6% and 71.9%.

Thus, landscape keyboard layout is also vulnerable. Moreover, this shows that the leakage contains more information than predicted by the leakage model of Section 2.3: had *only*

aggregate line intensities leaked, horizontally aligned keys would have been indistinguishable. We deduce there is exploitable information at a finer granularity than aggregate line intensities.

5 Text extraction

In this attack, the attacker aims to extract text from the attacked screen. This could be used to extract various types of sensitive data, such as identifying information, private messages, search queries, and financial information.

We assume that the attacker knows the content displayed on the attacked monitor, except for the textual letters. This assumption holds in many cases, for example, when the victim is filling a form on a known website. We also assume that the font is monospace and sufficiently large. The requisite size of the text depends on the granularity of the leak, which changes among different monitors. Another assumption is, again, that the screen is in portrait layout.

In this attack, we explore the possibility of extracting content from the screen rather than classifying it. We simulate an **open-world setting** where the word base rate is as low as 1/55000, i.e., a specific word has a 1/55000 probability of appearing in a trace. To simulate the low base rate, our implementation assumes that all characters and words are equally likely: it does not use any knowledge of actual word or character distribution. Notably, this is conservative for evaluating natural language word extraction, where the average base rate is much higher in practice.

Data collection and preprocessing. We captured traces of a Soyo 22" DYLM2248 in the close-range setting (see Section 3.1). We collected 10,000 traces, each 5 s long. In each trace, a different sequence of randomly chosen letters was displayed on the screen. The length of sequences was chosen randomly from $\{3, 4, 5, 6\}$. Letters were chosen from the English alphabet. Letters were capitalized, in font type Fixedsys Excelsior and size 175 pixels in width. Letters were black on a white screen.[7] We attach each trace to its corresponding sequence of characters, and again split the traces to train (90%) and validation (10%), and employ the same machine learning methodology as in Section 4.1.

Similarly, we also collected a test set: the traces of 100 English words, 25 for each possible length, chosen randomly similarly to Section 4.2.

We apply the signal processing algorithm described in Section 2.4.

Training procedure. We build one classifier for each character location. Each character location is rendered at a specific time segment during each refresh cycle. We match every character location with its time segment.[8] We construct training and validation data for each character by extracting the respective output trace segment, thus collecting pairs of output trace segment and corresponding character value.

We train the classifiers on the setup from Section 4.2.

CNN architecture: We used 1 convolutional layer with window size 12, 64 filters, and stride 1, one max-pooling layer with size 2, and 1 fully-connected layer with output size 2048. The

[7] In our modeling of this attack we display black text on a white screen. We expect the attack to generalize to when other content is surrounding the text, as long as the aggregate pixel intensity changes between different characters are prominent enough.

[8] To measure this, we introduced sharp pixel intensity changes in two different pixel lines and measured when, during the refresh cycle, these changes affect the signal. Because line rendering time is linear in line numbers, we can use this to construct an accurate mapping of lines to rendering time.

convolutional and FC layers have ReLU activations. The last layer is softmax. **Hyperparams:** We used an SGD optimizer with learning rate of 0.01, norm-clipped at 1. We used a batch size of 16, and trained for 1000 epochs. After every epoch the model was evaluated on the validation set; the best-performing model was saved for testing.

Testing procedure. We applied the following prediction scheme: for each of our 100 test set words, each classifier outputs a probability vector for each character. To predict which word was typed in, we use the Webster's dictionary [Web]. First, we format the words: we remove non-alphabetical characters, truncate or pad words with whitespaces so that they are six characters long, and then remove duplicates. This results in about 55,000 words. Second, we sum the log-probabilities of each word's characters. We then output the list of words, sorted by their probabilities.

Results. The per-character validation set accuracy (containing 10% of our 10,000 trace collection) ranges from 88% to 98%, except for the last character where the accuracy was 75%. Out of 100 recordings of test words, for two of them preprocessing returned an error. For 56 of them, the most probable word on the list was the correct one. For 72 of them, the correct word appeared in the list of top-five most probable words.

Error analysis. We found that when our attack predicts the wrong word, it often confuses a commonly used word for a rare one. For example, the word "dream" is erroneously predicted as "bream". To avoid such errors, we can introduce priors on word and character distributions (increasing our conservatively low base rate). For example, by collecting less randomized character sequences as training data or by assigning a higher probability for frequent words in the probability-assigning phase. If such changes were incorporated, accuracy would likely significantly increase, especially for commonly used words such as those in our training set.

6 Website distinguishing

In this attack, the attacker is interested in learning whether the victim is entering a specific website, or is just interested in the victim's website visiting habits. Website fingerprinting attacks, often studied in network traffic analysis settings [PLP$^+$, SST, CZJJ], convey the target user's browsing behavior and are very revealing of their hobbies and personality [KBK$^+$14]. In the VoIP setting, the attacker may be interested to learn in real time what the other parties to the call are doing. For example, he may wish to learn whether another party to the call is currently keeping the videoconference app maximized (presumably focusing his attention on the call), browsing Facebook, or responding to E-mails.

We note here that the classifiers described in Section 6.1 accurately distinguish up to 100 websites. Thus, as our methodology can accurately classify screen content, it can be leveraged for detecting an identified website visit vs. using common apps as well as classifying common apps. Finally, we also note that the collected website traces naturally contain dynamically-changing content (e.g., ads) affecting measurements, but classifiers nevertheles attain the high accuracy (implying that the general layout of a website is typically static and unique).

In Section 6.2, we directly evaluate a VoIP attacker whose goal is identifying the victim's foreground window, distinguishing between a VoIP app and popular websites.

6.1 Using the modulated signal

Data collection and preprocessing. We recorded traces of a Soyo 22" DYLM2248 at the close-range, at-distance, and phone attack settings (Sections 3.1 and 3.2). Out of the Moz top-500 website list [Moz], that ranks websites according to the number of other websites that contain links to them (which highly correlates with popularity), we chose 97 websites by filtering it for corner cases which may cause technical or other difficulties. For example, duplicate domains for the same website (e.g., `google.co.in` and `google.co.jp`) and non-US-based websites because some of them cause Selenium to hang.

We simulated the attack for the smartphone, at-distance, and close-range vantage points. For each, we iterated over the collection of websites over 100 times. In each iteration, we opened a Chrome browser window (controlled by the Selenium automation library for Python) with the URL of the website. We then started recording for 5 s. For each vantage point, we collected traces for 5 consecutive nights (when the recording machine was not otherwise in use). We stopped when we had reached 100 samples per website. This resulted in about 130 traces per website in the close-range vantage point, 100 traces from the at-distance vantage point, and 110 traces in the smartphone vantage point. For each setting, we used 70% of traces for training. For the close-range setting, we used the remaining 30% as a validation, which we used to guide our classifier architecture tuning (e.g., set learning rate, number of convolutional layers, etc). For the at-distance and smartphone settings, we used the remaining 30% as test sets.

We apply the signal processing algorithm described in Section 2.4.

Machine learning and results. Our task is to find which of the 97 websites is being displayed on the screen. We train a CNN directly to solve this task (see below CNN architecture), using the setup from Section 4.2. In about 8% of traces in the close-range and phone attacks, and about 16% in the at-distance attack, the signal processing algorithm returned an error, implying that the signal is particularly noisy. For the-close range setting, we attained a validation set accuracy of 97.09%. Then, when tested on the smartphone and at-distance test sets, the accuracy was 91.20% and 90.9% respectively.

CNN architecture: We used 6 convolutional layers, with a max-pooling layer after every 2. All convolutions are of window size 24 and stride 1. For the first and second layer, we have 16 and 32 filters respectively, and 64 for the other four convolutional layers. The 10th layer is a fully connected layer with 512 outputs, followed by a 0.9 dropout layer. The first 10 layers have ReLU activations. The last layer is an FC layer with softmax activations. **Hyperparams:** We used an SGD optimizer with a 0.01 learning rate, 0.1 gradient clipping, a categorical crossentropy loss, and batch size of 64. We trained each network for 800 epochs (about 4 s per epoch).

6.2 Attack through a Hangouts call

Here, we assume the attacker and victim are sharing a Hangouts video-chat session, where audio from the victim's environment is transmitted directly to the attacker. Leveraging acoustic leakage from the victim screen, the attacker's goal is to distinguish the scenario where video-chat is at the foreground from a scenario where the victim is surfing the Web, and also tell what website they are visiting.

Data collection and preprocessing. We recorded traces using the setup described in Section 3.3. We iteratively switched the foreground window of the victim screen in a round-robin fashion between 11 open windows: browser windows for each of the 10 top websites in Moz, and a (screen-shot) of a video-chat showing the human face of the attacker sitting in front of his webcam. We captured a 6 s recording before switching the next window into the foreground. In this way we collected 300 traces, 6 s each, for each open window.

Recall that in previous attack settings, we use the fact that the recurring pattern of interest is modulated and transmitted, in every refresh period, over a carrier signal at 32 kHz, and we leveraged this to produce a relatively clean version of a display-content-dependent leakage signal (see Section 2.4). Here, we can only sample at 44 kHz—below the Nyquist rate of the carrier signal. We can still, however, leverage the effect described in Section 2.3—namely, that pixel intensity directly affects the acoustic signal's amplitude. Traces were processed in a more straightforward way: we computed the fast Fourier transform of the 6 s trace. The resultant vector was used as input to the classifier. We split the traces to train (70%) and validation (30%).

Machine learning and results. Our task is to find which of 10 websites is being displayed on the screen. As in Section 6.1, we design a CNN directly trained to solve this task. The CNN reaches 96.5% accuracy on the validation set.

CNN architecture: We used 2 convolutional layers with kernel size 12, stride 1 and 32 filters, followed by a max-pooling layer with pool size 8, followed by a fully connected (FC) layer with output 512, followed by a Dropout 0.9 layer, followed by a softmax output layer. Convolutional, FC, and max-pooling layers have ReLU activations. **Hyperparams:** We used an Adadelta optimizer with categorical crossentropy and batch size 32. We trained for 100 epochs. Each epoch took about 36 s.

7 Cross-screen attack

In the experiments described thus far, training was done on recordings of the victim's screen. This simulates a scenario where the attacker has an opportunity to project images on the victim's screen and record the resulting audio (e.g., by physical access to the victim's screen, or during a videoconference call when the victim user is not watching). If such access is unavailable, the attacker could train on different screens of the same make and model (if known), or even completely different models. We now evaluate the attack's generalization to such cross-screen settings, where classifiers are trained on various collections of screens and then applied to another screen.

Data collection. We used a total of ten screens: five of model Dell 2208WFPt WD-04 (the *Dell4* set), two of model Dell 2208WFPt WD-05 (*Dell5*), two of model Dell 2208WFPf B (*DellB*), and one of model Soyo 22" DYLM2248. The screen set was chosen to contain both similarity (nine screens in the Dell 2208WFP* family) but also variation (including three different Dell models and one Soyo model). For every screen, we collected 50 recordings, 5 s each, of each of 25 websites (similarly to Section 6, the top 25 websites in the Moz top 500), in the close-range setting (see Section 3.1).

Data preprocessing. Firse, we applied the signal processing from Section 2.4. Then, for each victim screen v, we evaluate classifiers trained using each of the following "training collections":

- Each single screen (including v).

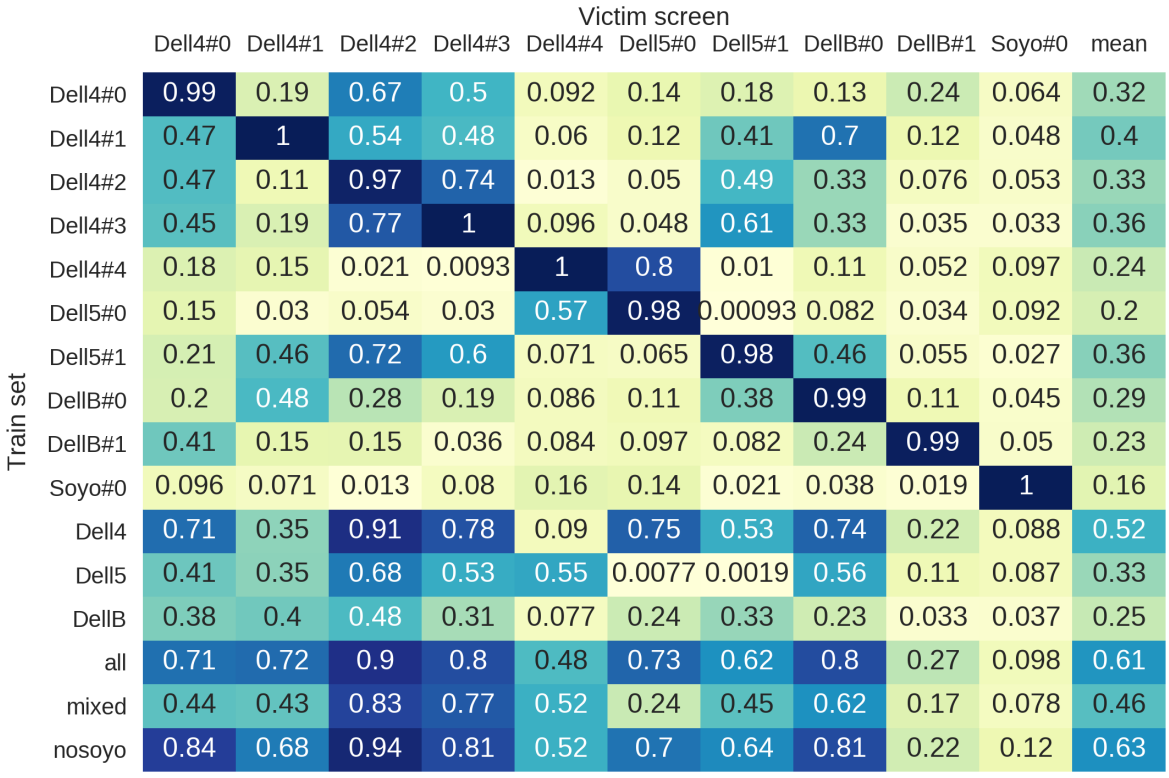


Figure 7.1: Cross-screen classification accuracy.

- Each of same-model sets (*Dell4*, *Dell5* and *DellB*) defined above, excluding v from the corresponding set.
- The *mixed* collection, containing 2 randomly chosen screens from *Dell4*, 1 from *Dell5*, and 1 from *DellB*, excluding v.
- The *all* collection, containing all 10 screens, excluding v.
- The *nosoyo* collection, with all screens except the Soyo.

For every such collection c, we assembled a training data containing about 50 samples for each website, equally distributed among the screens in c (i.e., we take $50/|c|$ samples for each screen in c).

Machine learning and results. We used our website distinguisher architecture in Section 6, but used the Adadelta optimizer which converges much faster than SGD when using only 25 classes, and thus trained for only 200 epochs. For each v, we trained our classifier on the training data of each of its training collections and evaluated the resulting classifier on v's data (re-initializing classifier learned weights at random before the next training process). The results are summarized in Figure 7.1.

Observations. First, we observe that classifiers trained on one of the 3 Dell models often generalized to a different model within the Dell 2208WFP* family. This sometimes happens even when training on just one screen (e.g., several classifiers generalize well on screen *Dell4#2*), and especially when training and testing on screens of the same model. Second, using more screens from the same family improves the intra-family generalization: training on a single screen yields worse results than training on two screens (i.e., *Dell4* or *Dell5*); training on 4

screens improves yields further improvement (*mixed* and *Dell4*), and training on 9-10 screens (*all* and *nosoyo*) gives the best results. Third, intra-model generalization is slightly higher than generalization between across the Dell models: for classifiers trained on a single Dell screen and tested a different screen of the same model, the average accuracy is 0.276, compared to 0.233 for screens of other Dell models. Finally, inter-vendor generalization is poor: classifiers trained on the Soyo screen have low accuracy for Dell screens and vice versa.

Conclusions. The accuracy of a remote attacker with no physical access to the screen is limited by inter-screen generalization. To attain high generalization, the attacker can use multiple screens from the same model, or even similar models from the same vendor. Note that this training phase can be done once, off-line, and utilized for multiple attacks. It can also be done retroactively, after recording the victim and using this recording to fingerprint their screen model.

8 Mitigations

Elimination. An obvious remedy to such leakage is for computer screen manufacturers to more carefully evaluate and implement their designs, aiming to minimize "coil whine" and similar vibrations of electronic components within the screen circuitry (cf. [GST]). The ubiquity of leakage, across manufacturers and models (Figures 2.3, 3.2, 3.3 demonstrate leakage in Dell, Soyo, Apple, Philips, HQ, BenQ, and Samsung; attacks were simulated on various Dell and Soyo screens in Sections 4 through 7), suggests that this may be difficult or costly.

Masking. Acoustic noise generators can be used to mask the signal, at extra cost in design, manufacturing, and ergonomic disruption (note that some of the exploitable signal lies within the human-audible frequency range). The masking ought to have adequate energy and spectrum coverage to reduce the signal-to-noise by orders of magnitude, because the image's leakage signal is retransmitted 60 times per second, offering high redundancy.

Shielding. Acoustic shielding of screens may reduce the amplitude of the leakage, but is difficult to achieve while keeping adequate air circulation for cooling. In the case of microphones built into screens, a sound-absorbing barrier around the microphone may reduce its pickup of internally-generated sounds (but would not affect external microphones). In the case of some screens or microphones in which there is an electromagnetic contribution to the leakage (see Section 2.2), corresponding shielding would also be desirable—and typically, expensive.

Software mitigations. A more promising approach to mitigating the attacks presented in this paper are software countermeasures. More specifically, variations on software mitigations to the EM Tempest attack, which change on-screen content to mask the leakage, such as font filtering [KA], can be considered. Moreover, because our extraction attacks use mainly aggregate horizontal intensity of pixel lines, while mostly losing the information inside individual lines, fonts can be changed such that all letters project the same horizontal intensity. Finally, our attacks all heavily rely on neural network classifiers which, themselves are vulnerable to inputs specifically crafted to mislead them [GSS14, SZS+13].

9 Conclusion

We report a new physical side-channel leak: acoustic leakage of screen content. We suggest *powerful attacks* that extract highly precise information, such as on-screen text and on-screen keyboard presses. We posit that this leakage is uniquely dangerous because even *weak attackers*, that only receive encoded audio traces from legitimate communication channels, as well as a attackers with no access to the victim's physical screen, can exploit it.

We demonstrate this by successfully simulating highly precise content extraction and identification attacks across an array of setups, as well as a simple, but well motivated, attack scenario: exploiting an open Hangouts connection to deduce what on-screen activity a party to the call is involved in.

This is the first demonstrated attack using codec-encoded acoustic emanations from non-mechanical peripherals, for which users don't have a reason to suspect acoustic leakage would even exist.

Acknowledgments

This work was supported by the Blavatnik Interdisciplinary Cyber Research Center (ICRC); by the Check Point Institute for Information Security; by the Defense Advanced Research Project Agency (DARPA) and Army Research Office (ARO) under Contract #W911NF-15-C-0236; by the Defense Advanced Research Project Agency (DARPA) under Contract #FA8650-16-C-7622. by the Israeli Ministry of Science and Technology; by the Leona M. & Harry B. Helmsley Charitable Trust; by NSF awards #CNS-1445424 and #CCF-1423306; by the 2017-2018 Rothschild Postdoctoral Fellowship; by the financial assistance award 70NANB15H328 from the U.S. Department of Commerce National Institute of Standards and Technology. Any opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect the views of ARO, DARPA, NSF, the U.S. Government or other sponsors.

References

[AAa] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy 2004*.

[AAb] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy 2004*.

[And08] Ross J. Anderson. *Security Engineering — A Guide to Building Dependable Distributed Systems (2nd ed.)*. Wiley, 2008.

[BB05] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[BCV] Davide Balzarotti, Marco Cova, and Giovanni Vigna. Clearshot: Eavesdropping on keyboard input from video. In *IEEE Symposium on Security and Privacy 2008*.

[BDG+] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. In *USENIX Security Symposium 2010*.

[BFMRT] Pierre Belgarric, Pierre-Alain Fouque, Gilles Macario-Rat, and Mehdi Tibouchi. Side-channel analysis of Weierstrass and Koblitz curve ECDSA on Android smartphones. In *RSA Conference Cryptographers' Track (CT-RSA) 2016*.

[BT] Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In *European Symposium on Research in Computer Security (ESORICS) 2011*.

[BWY] Yigael Berger, Avishai Wool, and Arie Yeredor. Dictionary attacks using keyboard acoustic emanations. In *ACM Conference on Computer and Communications Security (CCS) 2006*.

[CCLT] Alberto Compagno, Mauro Conti, Daniele Lain, and Gene Tsudik. Don't skype & type!: Acoustic eavesdropping in voice-over-ip. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS) 2017*.

[CMR+] Shane S Clark, Hossen Mustafa, Benjamin Ransford, Jacob Sorber, Kevin Fu, and Wenyuan Xu. Current events: Identifying webpages by tapping the electrical outlet. In *European Symposium on Research in Computer Security (ESORICS) 2013*.

[CRR+] Shane S Clark, Benjamin Ransford, Amir Rahmati, Shane Guineau, Jacob Sorber, Wenyuan Xu, Kevin Fu, A Rahmati, M Salajegheh, D Holcomb, et al. Wattsupdoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices. In *HealthTech 2013*.

[CZJJ] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *ACM Conference on Computer and Communications Security (CCS) 2012*.

[CZP] Robert Callan, Alenka Zajić, and Milos Prvulovic. Fase: Finding amplitude-modulated side-channel emanations. In *ISCA 2015*.

[EGKP] Miro Enev, Sidhant Gupta, Tadayoshi Kohno, and Shwetak N Patel. Televisions, video privacy, and powerline electromagnetic interference. In *ACM Conference on Computer and Communications Security (CCS) 2011*.

[EL85] Wim Van Eck and Neher Laborato. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security*, 4:269–286, 1985.

[FHE+] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *IEEE Symposium on Security and Privacy 2012*.

[GMO] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: concrete results. In *CHES 2001*.

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

[GPP+16] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Adi Shamir, and Eran Tromer. Physical key extraction attacks on PCs. *Communications of the ACM*, 59(6):70–79, 2016.

[GPT] Daniel Genkin, Itamar Pipman, and Eran Tromer. Get your hands off my laptop: Physical side-channel key-extraction attacks on pcs. In *Cryptology ePrint Archive, Report 2014/626*.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[GST] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *CRYPTO 2014*.

[Hah96] S.L. Hahn. *Hilbert transforms in signal processing*. Artech House Signal Processing Library. Artech House, 1996.

[HS] Tzipora Halevi and Nitesh Saxena. A closer look at keyboard acoustic emanations: random passwords, typing styles and decoding techniques. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS) 2012*.

[HS15] Tzipora Halevi and Nitesh Saxena. Keyboard acoustic side channel attacks: exploring realistic and security-sensitive scenarios. *International Journal of Information Security*, 14(5):443–456, 2015.

[KA] Markus G Kuhn and Ross J Anderson. Soft tempest: Hidden data transmission using electromagnetic emanations. In *International Workshop on Information Hiding 1998*.

[KBK+14] Michal Kosinski, Yoram Bachrach, Pushmeet Kohli, David Stillwell, and Thore Graepel. Manifestations of user personality in website choice and behaviour on online social networks. *Machine Learning*, 95(3):357–380, 2014.

[KJJ] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO 1999*.

[KJJR11] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

[Koc] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In *CRYPTO 1996*.

[Kuh] Markus G. Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In *International Symposium on Privacy Enhancing Technologies Symposium (PETS) 2004*.

[Kuh03] Markus G. Kuhn. Compromising emanations: eavesdropping risks of computer displays. Technical Report UCAM-CL-TR-577, University of Cambridge, Computer Laboratory, December 2003. Ph.D. thesis. URL: `http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-577.pdf`.

[LWK+] Jian Liu, Yan Wang, Gorkem Kar, Yingying Chen, Jie Yang, and Marco Gruteser. Snooping keystrokes with mm-level audio ranging on a single phone. In *International Conference on Mobile Computing and Networking (MobiCom) 2015*.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks — Revealing the Secrets of Smart Cards*. Springer, 2007.

[Moz] Moz. The moz top 500. `https://moz.com/top500`. Accessed: Oct 23, 2017.

[Nat] National Security Agency. NACSIM 5000: TEMPEST fundamentals. In *Cryptome*.

[Nor] David Norman. 1000 common english words. `https://gist.github.com/deekayen/4148741`. Accessed: Jan 2018.

[PLP+] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *Network and Distributed System Security Symposium (NDSS) 2016*.

[QS] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *Smart Card Programming and Security 2001*.

[SGRY] Yang Su, Daniel Genkin, Damith Ranasinghe, and Yuval Yarom. Usb snooping made easy: Crosstalk leakage attacks on usb hubs. In *USENIX Security Symposium 2017*.

[SST] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *USENIX Security Symposium 2017*.

[ST] Roei Schuster and Eran Tromer. DroidDisintegrator: Intra-application information flow control in android apps. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS) 2016*.

[Ste] StereoMatch. Amazing MP3 recorder. `https://play.google.com/store/apps/details?id=com.stereomatch.mp3.audio.recorder`. Accessed: Jan 2018.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[Tod18] USA Today. How to listen to what Amazon Alexa has recorded in your home. `https://www.usatoday.com/story/tech/talkingtech/2018/05/28/how-listen-what-amazon-alexa-has-recorded-your-home/649814002/`, 2018.

[VP] Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In *USENIX Security Symposium 2009*.

[Web] Noah Webster. Webster's unabridged dictionary. `http://www.gutenberg.org/ebooks/29765`. Accessed: Jan 2018.

[Wik] Wikipedia. Convolutional neural network. `https://en.wikipedia.org/wiki/Convolutional_neural_network`. Accessed: Jan 2018.

[ZZT09] Li Zhuang, Feng Zhou, and J Doug Tygar. Keyboard acoustic emanations revisited. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):3, 2009.

A Appendix

A.1 Trace chunking algorithm in detail

The algorithm first initiates an empty collection of chunks. Then, it searches for the first two consecutive chunks whose sizes are in G, and their correlation[9] is higher than the threshold. It sets one of these chunks to be the first, master chunk, and adds it to the collection.

Then, starting from the position after the added chunk, it searches a $size \in G$ for the next chunk such that the correlation of the *following* chunk is the highest. If the master's correlation with the obtained chunk is above T, it is added to the chunk collection. Otherwise, it is discarded and the algorithm goes into *sync mode*.

In sync mode, the next chunk's size can be between S and $S + 6000$ samples. Again, the algorithm finds the size that maximizes correlation with the following chunk. As long as the algorithm is in sync mode, it does not add new chunks to the collection. The algorithm exits sync mode once it found a size such that the following chunk has correlation $< T$.

When the algorithm succeeds, it truncates all chunks to the size of the smallest one (typically $min\{G\}$), and performs outlier rejection: discard the 10% of chunks whose correlation with the master chunk is lowest, as well as any chunk that has a peak that exceeds 1.5 times the average highest peak. The pseudo-code is given in Algorithm 1.

Algorithm 1: algochunk

```
1: procedure ChopSignal(array signal, size S, allowed drift d, threshold T)
2: init:
3:     chunks ← list()
4:     G ← [S − d − 1, S + d]
5:     c_len ← median{G} // expected chunk length
6:
7: find_first, master_chunk:
8:     while len(chunks) = 0 do
9:         j ← argmax_{j∈G}{corr(signal[: j], signal[j : j + c_len])}
10:        if corr(signal[: j], signal[j + 1 :]) > T then
11:            chunks[0] ← signal[: j] // master chunk
12:            signal ← signal[j :]
13:        else
14:            signal ← signal[1 :]
15: loop:
16:    next_len ← G //normal mode
17:    state ← "normal"
18:    sync_count ← 0
```

[9] Pearson correlation of two chunks of different sizes is taken after truncating them to the lower size among the two.

```
    while len(signal) > len(chunks[0]) + max{next_len} do
        j ← argmax_{j∈next_len}{corr(chunks[0], signal[j : c_len])}
        c ← corr(chunks[0], signal[j : c_len])
        if corr(chunks[0], signal[: j]) > T and state = "normal" then
            chunks.append(signal[: j]) // master chunk
        else
            if c < T then
                state ← "sync"
                next_len ← [S, S + 6000]
                sync_count+ = 1
                if sync_count > 3 then
                    return "error"
            else
                state ← "normal"
                next_len ← G
        signal ← signal[j :]
outlier rejection:
    outlier_reject(chunks)
    return chunks
```