Lend Me Your Ear: Passive Remote Physical Side Channels on PCs

Daniel Genkin
Georgia Tech
genkin@gatech.edu

Noam Nissan
Tel Aviv University
noamnissan@post.tau.ac.il

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

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

Abstract

We show that built-in sensors in commodity PCs, such as microphones, inadvertently capture electromagnetic side-channel leakage from ongoing computation. Moreover, this information is often conveyed by supposedly-benign channels such as audio recordings and common Voice-over-IP applications, even after lossy compression.

Thus, we show, it is possible to conduct physical side-channel attacks on computation by remote and purely passive analysis of commonly-shared channels. These attacks require neither physical proximity (which could be mitigated by distance and shielding), nor the ability to run code on the target or configure its hardware. Consequently, we argue, physical side channels on PCs can no longer be excluded from remote-attack threat models.

We analyze the computation-dependent leakage captured by internal microphones, and empirically demonstrate its efficacy for attacks. In one scenario, an attacker steals the secret ECDSA signing keys of the counterparty in a voice call. In another, the attacker detects what web page their counterparty is loading. In the third scenario, a player in the Counter-Strike online multiplayer game can detect a hidden opponent waiting in ambush, by analyzing how the 3D rendering done by the opponent's computer induces faint but detectable signals into the opponent's audio feed.

1 Introduction

Physical side-channel attacks utilize unintended interactions between computing devices and the physical universe. The past two decades have seen such attacks become powerful tools for adversaries seeking to extract otherwise-unavailable information. Exploiting a plethora of physical effects such as power consumption [46], electromagnetic emanations [2], timing [17], or acoustics [29], physical side-channel attacks have been used to attack cryptographic implementations [26, 47, 52, 63], screens [27, 49], keyboards [6] or even printers [7].

Authors are ordered alphabetically.

While powerful, traditional physical side channel attacks have a significant limitation: the signals are acquired by external, attacker-controlled measurement equipment. This requires the attacker to gain physical proximity to the target device for the duration of the attack. Consequently, physical side channel attacks on PC-class computers are often considered to be "outside of the threat model" (e.g., laptops are typically protected and observed by their owners, servers are typically located in protected data centers, and anyway physical access to PCs often enables easier attacks).

Recently, several works challenge the need for external equipment for physical attacks. For embedded devices, it was shown [30, 36, 61, 64, 66, 79] that an attacker with access to configurable hardware (e.g., FPGA oscillators, time-delay converters, or ADCs), can use these components for constructing side-channel sensors, thereby measuring physical emanations. These attacks only consider embedded systems, which are meant to provide low-level configurable hardware interfaces typically not present on PCs. For the PC-class devices, Platypus [50] leverages local code execution to monitor the CPU's power consumption via the RAPL interface on `x86` processors, enabling power-analysis attacks. Platypus assumes the presence of on-chip power monitoring, and was recently blocked by requiring root privileges to access RAPL.

We observe that PC-class computers, such as laptops, nowadays include various built-in sensors: microphones, cameras, WiFi and cellular interfaces, accelerometers, etc. While these are intended and expected to capture some types of information (e.g., the users' voice or screen orientation), the acquired analog signals may inadvertently be contaminated with EM fields within the computer, and might consequently be affected by ongoing computation and its secret arguments.

Thus, we pose the following question:

Can the built-in sensors of PC devices be exploited to conduct physical side-channel attacks entirely within software, without any external measurement equipment? If so, how can unprivileged attacks exploit such information to recover secrets within computations?

Second, all the above attacks assume that the attacker can

execute code on the target device. We observe that digitized analog signals from internal secrets are often willingly shared with others, e.g., voice and video images during Voice over IP (VoIP) calls. This raises the possibility of *passive* attacks, that do not directly access any sensors nor require local code execution, and only utilize analog signals naturally shared by common apps. We thus pose the following second question:

Can physical side-channel attacks on PCs be conducted through common channels established by benign applications such as VoIP calls, with no physical proximity nor local code execution? What would it take for potential attacks to mount such "passive" remote physical attacks?

1.1 Our Contributions

We present a new type of side-channel leakage from PC: computation-dependent leakage observed through the laptop's audio interface. We show that built-in microphones in laptop computers, while designed to record human voices, are often capable of inadvertently recording computation-dependent electromagnetic (EM) leakage, allowing physical side channel attacks on PCs by purely software means. We show several exploitation scenarios:

Signal acquisition over VoIP. The computation-dependent EM leakage present on the laptop's audio interface is picked up and conveyed by multiple VoIP applications, despite their lossy audio compression codecs. Thus, the acquisition of the target's EM emanations can be done remotely, via a VoIP audio conversation over the Internet, without any physical access, proximity, or code execution on the target machine.

Web browsing activity. Demonstrating the feasibility of remote software-based physical attacks on PCs, we extract information about the victim's web browsing habits, via the audio signals available on the attacker's side of a VoIP conversation with the target. We empirically identify the website rendered by the target's browser, from among a set of Alexa's top-30 most popular websites. Further, we show that ambient noise in typical home and office settings has little effect on attack success, demonstrate that distinguishing information exists even in the presence of varying system conditions such as background processes, and discuss potential limitations such as user activity during the attack.

Cryptographic secrets. Next, we demonstrate that this signal can be exploited for remote extraction of cryptographic secret keys. Specifically, we extract 521-bit ECDSA signing keys from Libgcrypt 1.8.4, by analyzing leakage signals naturally sent by the target during a VoIP call with the attacker, while performing signing operations.

Gaming advantage. Finally, we show that this side channel can be used to cheat in on-line video games, by detecting the computational pattern of competing players' computers over an open VoIP call. Here, we demonstrate how in a first-person shooter (i.e., Counter-Strike), a hidden opponent waiting in ambush can be detected, avoided and sneaked up upon.

Summary of Contributions. To summarize, in this paper we make the following contributions.

- We show that CPU computation leaks onto audio signals recorded by a laptop's own microphone via commonly used APIs, across multiple laptop models and makes (Section 2).
- We show that the leakage is naturally transmitted via VoIP communication of common applications (Section 2), allowing for remote exploitation.
- We exploit this leakage to distinguish between websites open in the target's browser (Section 4).
- We demonstrate secret-key extraction from Libgcrypt's ECDSA implementation (Section 5).
- We demonstrate how to leverage this side channel to cheat on the popular video game Counter-Strike (Section 6).

1.2 Related Work

Remote timing side channels. Timing attacks exploit the correlation between the computation's execution time and its inputs, allowing attacker to recover secrets by merely observing the running time of the targeted operation. Such attacks have been demonstrated to be able to recover private RSA keys [17, 48], ECDSA signing keys [16] as well as violate security guarantees by TPM modules [42, 54].

Microarchitectural side channels. Microarchitectural attacks exploit the minute timing variations that result from contention on internal CPU resources in order to extract secret information. Assuming a co-location between the attacker's code and the target on the same physical machine, microarchitectural attacks have been shown to exploit caches [56, 58, 78], branch predictors [1, 25] pipeline components [4, 5, 37]. More recently, microarchitectural attacks have been extended to include speculation, breaking nearly all hardware based security domains [19, 45, 51, 72, 75, 76].

Physical side-channels attacks. As mentioned above, electronic computing devices also leak secrets through several physical side channels, including electromagnetic (EM) radiation [2, 24, 26, 49, 63, 74], acoustic emanation [29] and power consumptions [21, 46, 47, 52]. More recently, [28, 29] showed that PC devices leak via EM side channels. Finally, "Screaming channels" [18] showed that mixed signal system on chips (SoCs) inadvertantly transmit information on a processor's activity on top of a radio signal, thus allowing key extraction at a relatively far range of to up to 10 meters.

Software-based physical side channels on embedded devices. Analog-to-digital converters (ADCs) used by embedded hardware to digitize analog inputs were shown to pick up leakage of information on the execution of a processor located on the same chip [34, 57]. A recent line of work explores the possibility of repurposing FPGA fabrics as power monitors, aiming to pick up side-channel leakage from nearby circuitry. This approach can extract cryptographic keys from neighboring FPGAs [32, 64, 66, 79] or CPUs [36, 79] that share a chip with the attacker one, across different chips that

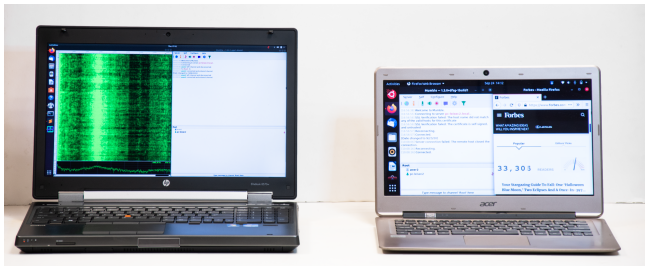


Figure 1: Experimental setup example. Attacker laptop (left) and victim laptop (Acer Aspire S3, right). Attacker and victim are both connected, via WiFi, to a Mumble VoIP chat room session. The green spectrogram on the attacker conveys the audio stream received from the victim.

share a board [67], or even via a covert channel across boards that share a power supply unit [65]. Power signal leakage was shown to be measurable using commodity [60, 64, 67, 79], cloud-based [31, 33], and multi-user FPGAs [31, 35, 60].

Software-based physical side channel attacks on complex devices. Software-based physical side channel attacks are not entirely exclusive to embedded devices. Indeed, Platypus [50] shows how an attacker with local code execution can use the RAPL interface on Intel and AMD processors to monitor the CPU's power consumption. Until blocked by OS vendors, this access allowed attackers with code-execution privileges to mount software-based power-analysis attacks on CPUs, breaking cryptographic implementations, derandomizing kernel ASLR, and leaking data from SGX enclaves. Software-based fault attacks were also demonstrated on SGX enclaves in Intel CPUs [44, 55] as well as on ARM TrustZone [62, 69].

Acoustic attacks on peripheral devices exploit the sound generated by mechanical elements of peripherals such as keyboards [6, 8, 10, 22, 38, 39, 77, 80], printers [7], and computer screens [27]. Our work differs in that it analyzes signals that stem from complex CPU computation rather than simple peripheral I/O. Further, our signals are not truly acoustic (they include EM emanations captured by microphone circuitry), are very faint (unlike mechanical noises), and are not trivially repetitive (unlike screen signals).

2 Characterizing the Leakage Signal

2.1 Observing Local Audio Leakage

To establish leakage through the target's own audio interface, we wrote a test program that repeatedly executes a 1 sec loop of integer multiplications, followed by a 1 sec loop of memory accesses, with short `sleep`s inbetween. We ran this program and recorded the target's microphone using its internal audio interface, sampled at 48 Ksample/sec, 16 bits per sample.

Figure 2 shows spectrograms of such recordings across 6 laptops of different models and makes (see Table 1 for details). While signal quality varies, all spectrograms show frequency alternations at 1-second intervals, corresponding to heavy-light CPU-load periods. We deduce that the integrated audio interface on commodity laptops captures side-channel information on CPU computation.

Model	Mfg. date	CPU	RAM	Sections
Acer Aspire S3	11/2011	i5-2467M 1.60 GHz	4GB	2,2.3, 4.1, 5
Acer Aspire S3	08/2012	i7-3517U 1.90 GHz	4GB	4.2
Acer Aspire S13	05/2016	i5-6500U 2.5 GHz	8GB	2,4.2
Acer Aspire E5	07/2018	i3-8130U 2.2 GHz	4GB	4.2
Lenovo IdeaPad Z370	mid 2011	i5-2410M 2.3 GHz	4GB	2
Lenovo ThinkPad X270	02/2018	i7-7500U 2.7 GHz	16GB	2
Lenovo ThinkPad X1 Carbon Gen 6	03/2018	i7-8650U 2.1 GHz	16GB	2
Lenovo Yoga 730-13IWL	04/2019	i5-8265U 1.8 GHz	16GB	2

Table 1: Target machines used in this paper

2.2 Observing Leakage via VoIP

Having established that CPU-computation information leaks onto a laptop's own microphone, we proceed to explore whether such signals can be acquired remotely over common VoIP applications. Such applications may reduce the fidelity of the leakage signals, since they often employ lossy audio codecs that are psychoacoustically optimized for speech and music, rather than faint and "unnatural" electronic noise.

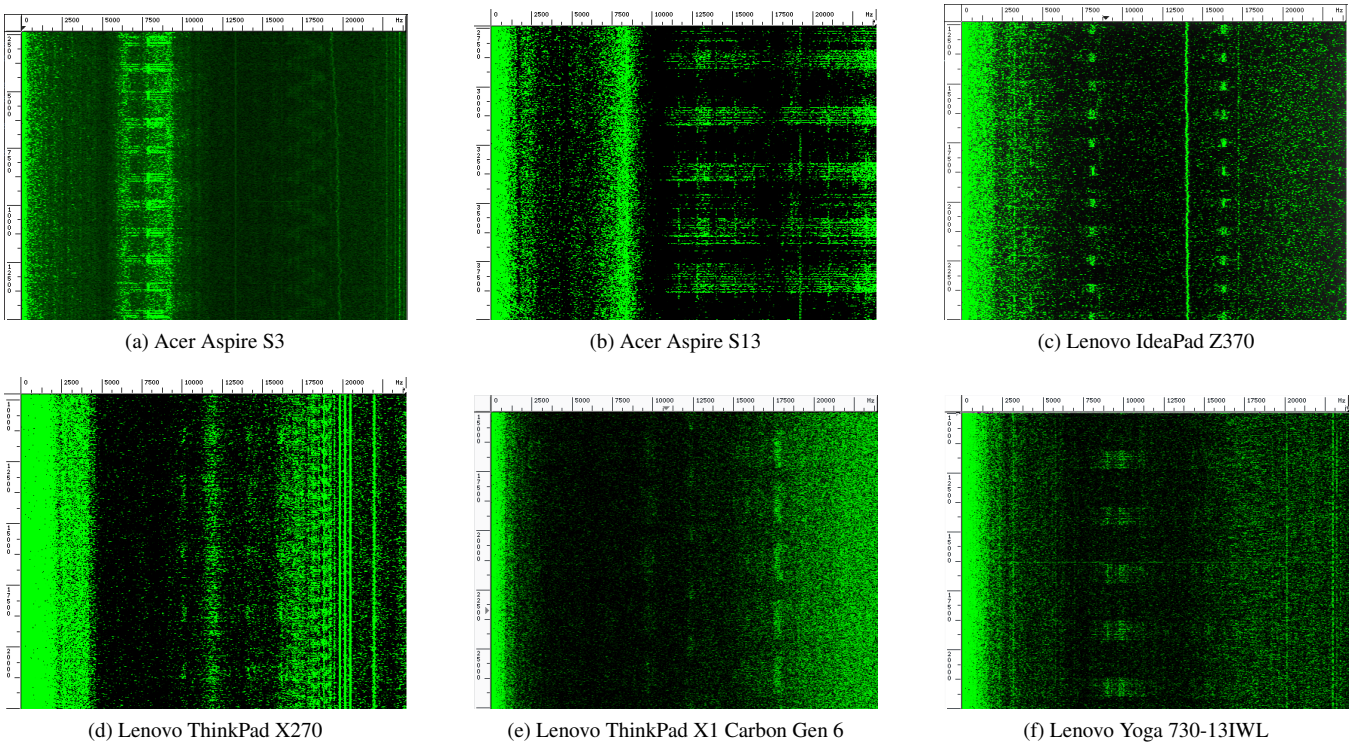
Experimental setup. Testing the feasibility of this scenario, we use the Acer Aspire S13 laptop from Figure 2b as a target, and setup a VoIP call between the attacker's machine and the Acer laptop, using 5 popular VoIP applications: Skype, Zoom, Google Meet, Mumble, and Discord. We keep the applications' default codec choice and compression parameters. All five of the tested VoIP applications use *Opus* [70] as their audio codec, which is considered to be the de-facto standard codec for real time communication, though each application uses a slightly different configuration of this codec.

During each call, we repeat the measurements performed in Figure 2, having the target alternate between CPU-heavy and memory-heavy workloads. On the attacker's machine, we capture the remote participant's (i.e., target's) audio through a VoIP application, via Linux's audio loopback interface.

Remotely observing the leakage signal. Figure 3 illustrates our findings. While quality varies, the signals generated by our test programs are clearly visible in all 5 applications, albeit with a reduced fidelity due to different low-pass filters, aggressive compression, and other processing.

2.3 Determining the Leakage Source

Having established the presence of computation-dependent signal on the laptop's audio interface, and that it is transmitted even to remote VoIP parties, we aim to identify the physical source of this leakage.



(a) Acer Aspire S3 (b) Acer Aspire S13 (c) Lenovo IdeaPad Z370

(d) Lenovo ThinkPad X270 (e) Lenovo ThinkPad X1 Carbon Gen 6 (f) Lenovo Yoga 730-13IWL

Figure 2: Distinguishing between heavy and light CPU loads in recordings of microphones embedded in various laptops. In these spectrograms, the vertical axis is time in milliseconds (≈12.5 sec total), and the horizontal is frequency (0–24 kHz). The laptop's lid is closed in Figure 2d, partially closed in Figures 2e and 2f, and open in the rest.

Acoustic or EM? As microphones are designed to be good audio receivers, one hypothesis is that the microphone somehow picks up acoustic emanations from internal circuitry on the machine's motherboard [29]. Alternatively, the ADC within the audio interface somehow picks up conducted or radiated EM emanations, sampling it as if it were originating from the outputs of the internal microphone. This may happen due to EM emanations affecting the transducer, signal-conditioning electronics (amplifiers and filters) or sampling electronics (A2D); or due to conducted fluctuations in the ground or power-supply rails of these electronics.

Experimental setup. To test whether the signal is acoustic or EM, we measured the aforementioned Acer S3 using three different sensors, simultaneously: (1) the laptop's embedded microphone sampling at 48 Ksample/sec, (2) an EM probe (Langer LF-R 400 connected to Tracker Pre sound card, sampling at 48 Ksample/sec), and (3) a consumer-grade microphone. The latter two use external equipment which is not connected to the target. Figure 4 shows the combined setup.

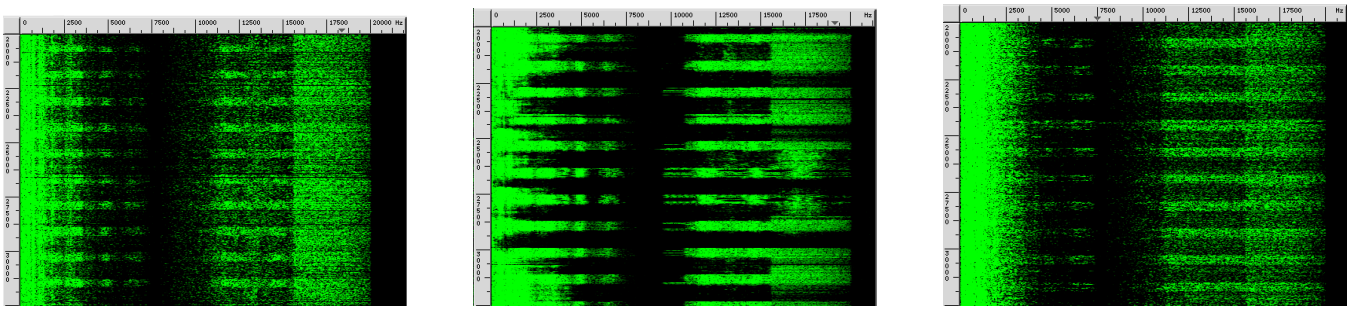
Observing the leakage signal. Figure 5 shows the signal measured via the internal audio interface, and the signal captured by the EM probe. They exhibit similar copmutation-dependent leakage (alternating bright and dark areas at around 7.5 kHz marked with red arrows, corresponding to memory and CPU operations performed by our test program). The external-microphone signal does not appear to contain this information. We deduce that the leakage is essentially electromagnetic, and is picked up by the analog front-end of the audio interface of the target machine, either via conduction or via radiation.

Locating the leakage source. To spatially localize the leakage source inside the laptop, we visualized the signal from our EM probe on a spectrogram, while moving it around near the laptop's chassis. Though leakage prominently appears in several locations, we find the strongest signal near the laptop's Escape key, in close proximity to both the internal microphone and to the CPU's voltage regulation circuits (see Figure 6). We thus conjecture that the EM signal is emitted by the CPU's voltage regulator, and is then picked up by the analog front-end of the laptop's internal audio interface.

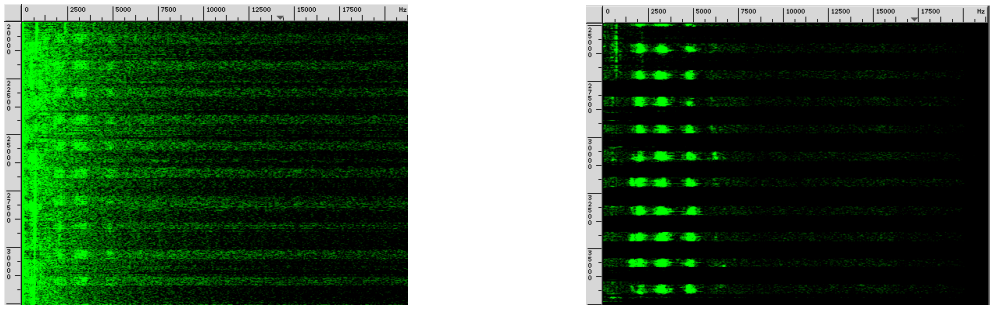
3 Threat Model

Our key assumption is that the attacker can remotely acquire digitally-recorded audio, captured by the victim device's microphone, while the victim is performing sensitive operations (e.g., website browsing, cryptographic signing, or game rendering). Such audio may be transmitted, e.g., via VoIP, video call, or webcasts.

Our experiments on website identification (Section 4) and



(a) Discord (b) Mumble (c) Google Meet



(d) Skype (e) Zoom

Figure 3: Signals from an Acer Aspire S13's embedded microphone, recorded by the remote party in VoIP and video-chat calls using various applications. The vertical axis is time (≈12.5 sec total), and the horizontal axis is frequency (0–22.5 kHz).

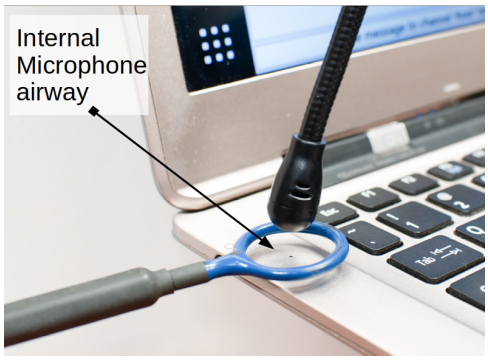


Figure 4: Measuring the Acer laptop with an EM probe (blue ring), internal microphone (black dot below blue ring) and external microphone (black rod above the blue ring).

key extraction (Section 5) make some additional assumptions that are not necessarily inherent to the attack: an ability to acquire multiple training traces from the victim's own device, and that the victim's website-visit patterns and conditions are highly regular. We discuss those extensively in Section 8.

Assumptions we *do not* make. Unlike most prior physical-side-channel attacks, we do not assume that the attacker has physical access or dedicated measurement equipment in proximity to the target. We do not assume that the attacker has code-execution capabilities on the target device, neither intentional nor via a code exploit.

We also do not assume that the audio signal obtained by the attacker is free from interference. Our attacker's audio recordings contain environmental acoustic noise such as music and human speech, internal acoustic noise such as that from the cooling fan, and interference by other software running on the target machine (including the operating system's background tasks, the VoIP application that connects the victim and attacker, and other processes).

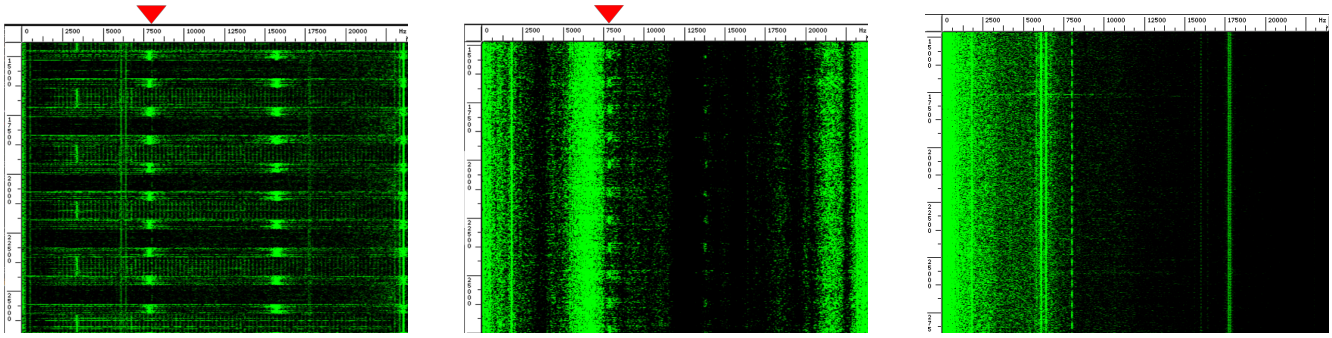
4 Website Identification

4.1 Distinguishing Animated News Websites

Having established and characterized the computation-dependent signal on the laptop's audio interface, we next show that this leakage can be used to identify websites visited by the victim. We begin this investigation with websites that display dynamically-changing on-screen content or videos, since the animation's continuous rendering may leave its footprint on our leakage signal across the entire span of time when the website remains displayed by the browser.

We show that leakage signals, captured by a laptop's built-in microphone and acquired by the remote party to a VoIP call, appear consistent across visits to the same website, and different across visits to different websites. A convolutional neural network (CNN) can identify which of 14 popular news websites is being visited with 96% accuracy.

Experimental setup. We used an Acer S3 running Ubuntu and a Firefox browser (in their default configuration) as the victim laptop. The target and attacker were connected over



(a) EM probe (b) Self measurement (c) External microphone

Figure 5: The trace from the laptop-embedded microphone in Figure 5b has similar features to the one acquired by an electromagnetic probe in Figure 5a, which are not present when measuring with an external microphone in Figure 5c.

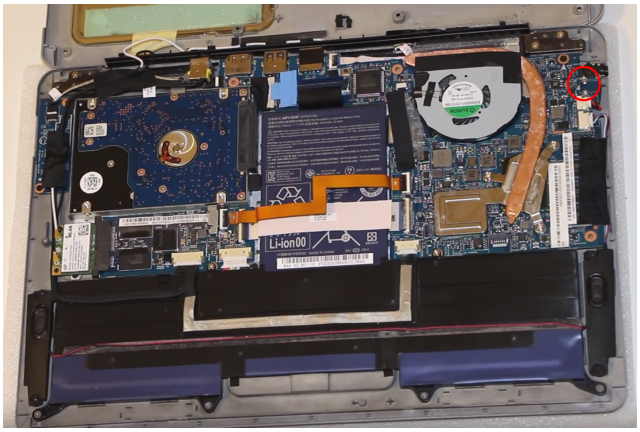


Figure 6: Proximity of the microphone to the CPU's voltage regulator circuit. (Photo from youtu.be/IO9YXLW7FdA.)

WiFi to a home router.We established a VoIP connection using Mumble (version 1.4.1), a popular open-source VoIP application. A Mumble server ran on the attacker machine, and both the victim and attacker connected to it for the duration of the experiment. We switched Mumble's transmission setting from "voice detection" to "continuous", and set the voice quality to maximum (96 kbit/sec). On the attacker's side, we captured Mumble's playback of the target's transmitted audio by recording the local audio loop-back interface, and used the recordings for further processing (described below).

Observing websites leakage signal. Figure 7 presents spectrograms of the victim's audio stream while browsing different websites, as received by the attacker over Mumble. The animated content on the websites created very different visual patterns (e.g., black and green stripes) on the spectrogram. These patterns were consistent across visits, and corresponded to the animations (marked in red) on the corresponding website. Computing the FFT spectra of the multiple traces of each of the three websites (Figure 7, bottom) we can see clear and consistent differences in several frequency bins.

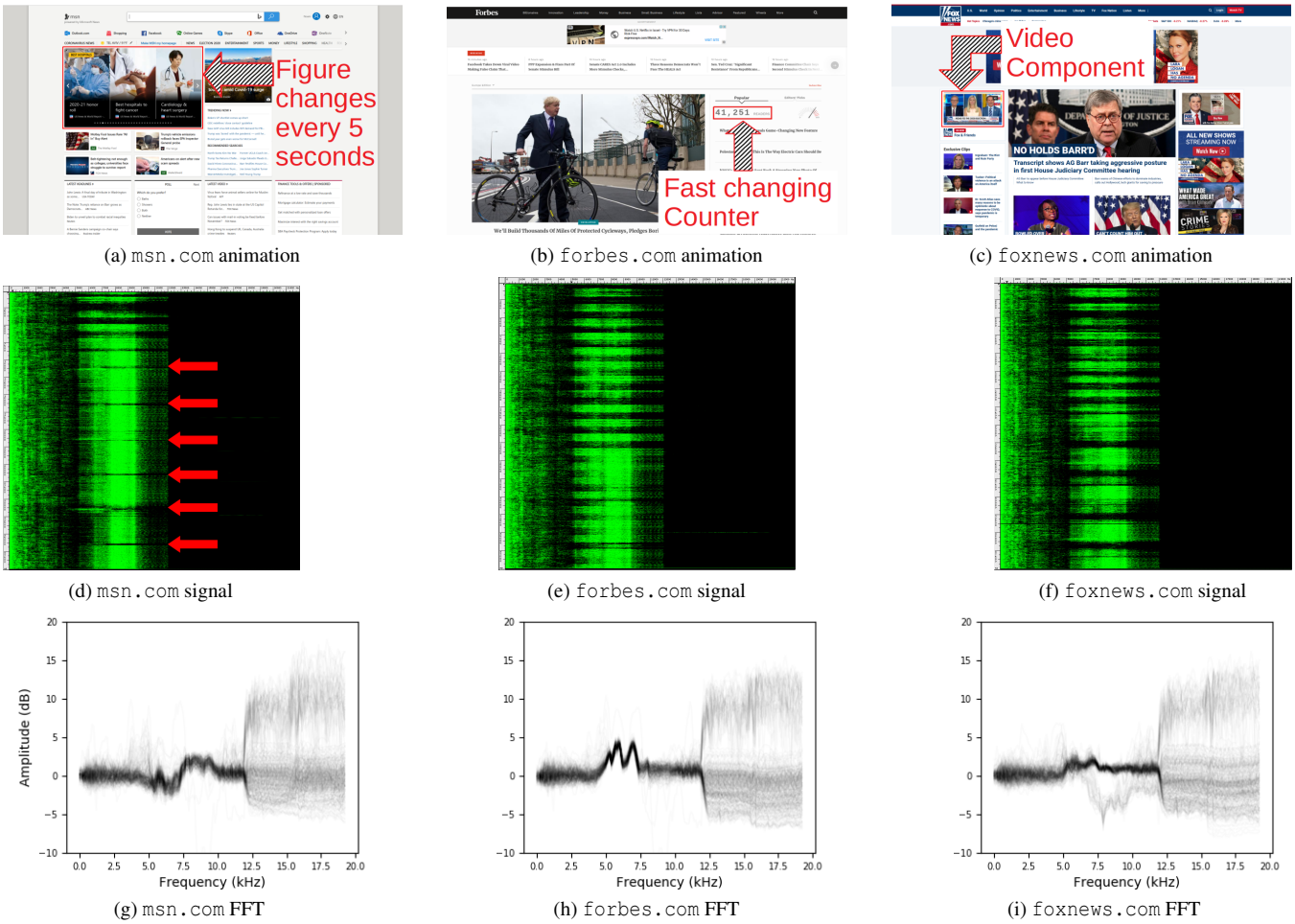
Data collection. We now aim to perform website distinguishing automatically and at a larger scale. To this end, we collected data using 14 websites from Alexa's list of most popular news websites where animation is displayed.[1] We used the Acer S3 laptop with the aforementioned VoIP setup, and opened each of the websites in a round-robin fashion. For each visit, we used Selenium's browser-control automation to launch a Firefox window directed to the website's address, left it open for about 40 seconds, and then closed it. We then allowed a 5-second sleep period before opening the next page. On the attacker end, we recorded the incoming audio during the time the page is open in the victim. We repeated this process for 250 rounds, generating a total of 3,500 samples.

This setup renders the victim's hardware/software conditions fairly consistent across the website visits recorded by the attacker. In Section 4.4 we measure the effect of varying background processes across visits, and in Section 8 we discuss how to address real-world victims who may further deviate from these laboratory conditions.

Signal preprocessing. For each collected signal, we first extracted a spectrogram using matplotlib's [41] `specgram` function from the `mlab` module. We used an FFT window size of 2048, and a Blackman window function with a signal overlap of 0.12. We computed the log (base 10) of each entry and scaled down the resulting spectrogram image using scikit-image's [73] `transform.resize` method, such that both dimension sizes (time and frequency) are divided by 5. We then cropped the image to filter out the first 10 seconds, which contain the loading of the website.

Machine learning methodology. After preprocessing every sample as above, we split the samples into a training set (85% of samples) and a test set (15%). To normalize, we divided all values by the highest value observed in he training set. We then trained a CNN classifier. CNNs have shown to be effective for time-series data acquired via a side channel, and particularly for audio, as they are comparatively accurate,

[1] Specifically: `reuters.com economictimes.indiatimes.com indianexpress.com news.com.au chainadaily.com.cn latimes.com nytimes.com forbes.com newsweek.com bloomberg.com/middleeast foxnews.com abcnews.go.com euronews.com nationalgeographic.com`



(a) `msn.com` animation (b) `forbes.com` animation (c) `foxnews.com` animation

(d) `msn.com` signal (e) `forbes.com` signal (f) `foxnews.com` signal

(g) `msn.com` FFT (h) `forbes.com` FFT (i) `foxnews.com` FFT

Figure 7: Attacker's signal when victim viewed `msn.com`, `forbes.com` and `foxnews.com`. For spectrograms, the vertical axis is time (40 sec), and the horizontal axis is frequency (0–24 kHz). The bottom figures show, for each website, the FFTs of all recordings in a single plot. We subtracted from the measurement the average value, over all websites, of each bin.

robust to noise, and tend to generalize across settings and devices [27, 68].

Our neural network includes 2 convolutional layers of dimensions 4×4 and with 8 and 16 filters respectively, followed by a 2×2 max-pooling layer, followed by a fully-connected layer with 256 outputs, followed by a linear layer with dimensions 256×14, followed by a soft-max operation. The first convolutional layer does a 4×4 stride. The convolutional and fully-connected layers are followed by ReLU activations. We used an Adadelta optimizer with learning rate 1, and batch size 32, and trained for 100 epochs.

For programming, training, and running our classifier, we used the deep learning library Keras 2.3.0 with Tensorflow 2.0 and CUDA 10.0 / CuDNN 7.4 backends. Training took about 90 second on using Intel i7-4930K CPU and NVIDIA GeForce RTX 2080 Ti GPU.

Results. The above model succeeded with value accuracy of 96%. Figure 8 shows the classifier's confusion matrix. It appears that variation in loading time (unusually slow or fast loading) were the main cause of errors.

4.2 Evaluating on Alexa Top Websites

In Section 4.1, we evaluated our attack on websites that display animations using the Acer S3 laptop. We now evaluate a website-classification adversary on Alexa's unfiltered top website list, not all of which contain animations, using also newer hardware. We applied a similar methodology, but enhanced trace spectral accuracy by averaging together rows of the spectrogram (i.e. local spectrum) corresponding to times where the CPU is active, as follows.

Enhancing spectral accuracy. As some of the websites do not contain animations, temporal patterns of animation rendering are not as discernible to the naked eye as those in Figure 7. Yet, we expect the ongoing rendering of websites, and execution of their internal scripts, to leak information about their

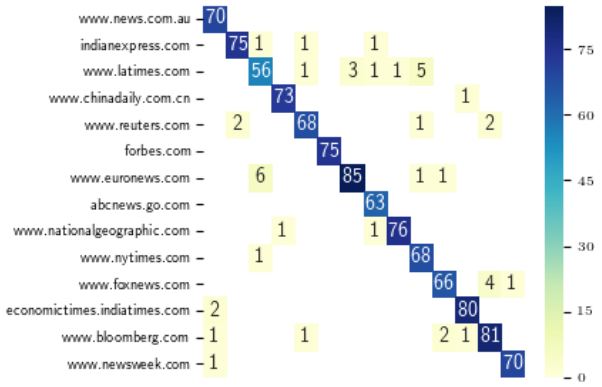


Figure 8: Confusion matrix of the website classifier.

content onto the signal spectrum. We observe our signals contain regular periods of stronger activity, interspersed with relatively quiescent periods (Figure 7). We conjecture that stronger periods are ones where the CPU is busy (and rendering the website), whereas silent ones correspond to idle times (which presumably contain no useful information, as the website is not being rendered). We filtered out idle times by (1) choosing the maximal frequency in the signal's real FFT spectrum between 5 kHz and 12.5 kHz, whose values are (we found) indicative of CPU activity, (2) computing the series of values in the spectrogram bin corresponding to the chosen frequency, and (3) selecting times where it is above its median value. Then, we computed the FFT spectra of the signal during the selected times. Finally, we normalized each trace by dividing all values by the maximal absolute value observed in the dataset, and subtracted the average of all traces. This resulted in a 2048-dimension vector for each trace, which we used as input to our machine-learning classifier.

Experimental setup. We used an Acer S3, an Acer S13, and an Acer E5 as our victims, and collected traces as in Section 4.1 from Alexa's top website list from November 2020. We used 350 30-second traces for each website, this time without discarding the first 10 seconds. We split the dataset to train and test sets as in Section 4.1 and trained a neural network for 1000 epochs on the training set.

Our neural network includes a single 1D convolution layer with kernel size 4 and 32 filters, followed by 2-way max-pooling, a 128-neuron fully connected layer, followed by a softmax layer. Again, we used ReLU activations at all hidden layers, and an Adadelta optimizer with learning rate 1.

Results. Table 2 summarizes our results for attempting to distinguish among the 15 and among the 30 top Alexa websites. While the accuracy is lower than in Section 4.1, where animation-rendering times contributed to website distinguishability, it is far above a random-guessing baseline (6.6% for 15 websites or 3.3% for 30 websites).

Victim model	website count	accuracy	CPU model	manufacturing date
Acer S3	30	40%	i7-3517U	08/12
Acer S3	15	50%	i7-3517U	08/12
Acer S13	15	69%	i5-6500U	05/16
Acer E5	15	65%	i3-8130U	07/18

Table 2: Results: Alexa top websites. Accuracy is the correct-classification rate for a 30-second trace.

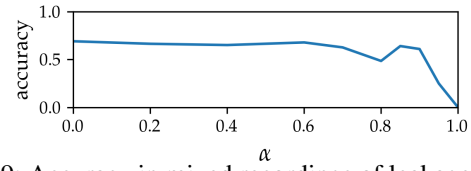


Figure 9: Accuracy in mixed recordings of leakage and *The Office*, weighted $1-\alpha$ and α respectively.

4.3 Effect of Ambient Noise

Due to the COVID-19 pandemic, the recordings in Section 4.2 were taken within the corresponding author's living room, resulting in the microphone of the target laptop recording ambient noises from music, speech, TV shows, and other noises resulting from human activity. Nonetheless, website identification is still possible, as shown in Table 2.

To more systematically evaluate the performance of our technique in the presence of different noise levels from a realistic office environment, we augmented our recordings by mixing them with audio from the TV show *The Office*. We used episodes from seasons 1 and 2, totaling $\sim$ 10 hours of playback at the same sample rate as our traces. Using the Acer S13 dataset from Section 4.2, we mixed each trace with a 30-seconds segment of *The Office*, chosen at a random offset from a random episode. Mixing was done via a weighted average between the office noise and the leakage signal, with weight α given to the office noise, and weight $1-\alpha$ given to the signal, for various values of $\alpha \in [0,1]$. For each α, we separately applied the methodology of Section 4.2. Figure 9 depicts the result. Even though audible sound is dominated by *The Office*'s playback already for $\alpha \geq 0.2$, accuracy starts to significantly decrease only for $\alpha \geq 0.9$. Thus, the attack is able to recover website information even when the leakage signal's amplitude is attenuated tenfold and masked by much stronger ambient noise.

4.4 Noise From Other Software Processes

Software activity on the target machine is another source of noise in the attacker's signals. Our experiments were conducted on a standard installation of the OS, including all of its background processes. Additional load was induced by the VoIP software on the target machine, and any auxiliary tasks executed by the web browser.

Additional applications might induce further interference. For interactive applications, which are mostly dormant while awaiting user input, this effect is usually minor. Figure 10

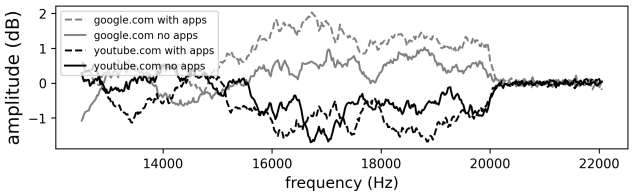


Figure 10: Spectra of 5-minute traces for google.com and youtube.com (Alexa's top-2 sites) as output by our procedure from Section 4.2, with and without background applications (Thunderbird Mail, Spotify, LibreOffice Writer, and Signal messenger) running.

illustrates this, showing how spectra of traces recorded while browsing youtube.com, are distinct from those recorded with google.com, even with multiple applications running.

5 Cryptographic Key Recovery

Having established the feasibility of website classification using the laptop's audio interface, we now demonstrate that the audio-interface leakage can be used to extract secret keys from cryptographic computations. Specifically, we show an attack on the ECDSA-signing implementation of Libgcrypt 1.8.4. We exploit the same side-channel weakness as Minerva [42], however, instead of leveraging measurement code locally running on the target machine, we perform a remote attack over the Internet, using only the audio information transmitted during a Mumble VoIP session.

5.1 Mathematical Background

We provide necessary mathematical background on ECDSA, and its deterministic-k variant, as implemented by Libgcrypt.

ECDSA. The Elliptic-Curve Digital Signature Algorithm (ECDSA) comprises three functions: key generation, signing, and signature verification. Given an elliptic curve group of order n with generator $\mathbb{G}$, key generation is done by having the signer generate a random integer $d \in [1, n-1]$ and compute $\mathbb{Q} = [d]\mathbb{G}$ (where $[d]\mathbb{G}$ is the scalar-by-point multiplication inside the group generated by $\mathbb{G}$). Finally, the integer d is the secret signing key and $\mathbb{Q}$ is the public verification key.

Next, to sign a message m, the signer first hashes m, obtaining z by truncating the resulting digest into $\lceil \log_2 n \rceil$ bits. The signer than samples a random secret nonce k, and computes the curve point $(x,y) = [k]\mathbb{G}$, $r = x \mod n$ and $s = k^{-1}(z + r \cdot d) \mod n$. The output signature is (r,s). To verify a signature (r,s) corresponding to a message m with a truncated hash digest z, the verifier computes $w = s^{-1} \mod n$, $u_1 = zw \mod n$, $u_2 = rw \mod n$, $(x,y) = [u_1]\mathbb{G} + [u_2]\mathbb{Q}$ and checks that $x \equiv y \mod n$.

ECDSA with deterministic nonces. The requirement that the nonce k be chosen uniformly at random makes ECDSA hard to implement securely and to test. Biases and implementation weaknesses in the random generation of k are often mathematically exploitable by tools such as lattice reduction, and have led to numerous vulnerabilities in deployed systems [13–15, 23, 40].

To mitigate this issue, RFC 6979 [59] (natively supported by Libgcrypt) suggests a method where the nonce k is generated deterministically from m and d, using a series of HMAC applications. In ECDSA with deterministic nonces, k is uniquely and deterministically determined from m and d. The idea is that, despite this determinism, RFC 6979 [59] is ostensibly secure: as long as the attacker does not know d, they are unable to predict k. Ironically, however, this mitigation actually aggravates side-channel weaknesses, as it allows attackers to amplify their signal-to-noise by using multiple traces of signing the same message (thus having the same nonce). Real-world applications often repeatedly sign the same data, e.g. when signing identical files or generating S/MIME-signed autoresponse, making such attacks dangerous.

Scalar-by-point multiplication. To avoid secret-dependent runtime variations, Libgcrypt 1.8.4 computes $[k]\mathbb{G}$ using a left-to-right double-and-always-add methodAlgorithm 1. The bits of k are processed in one main loop which performs a single double and a single add operation in each iteration (Lines 5 and 6). As the result of the add operation is only needed in case the current bit is 1, a constant-time conditional swap is used to update Q when $k_i = 1$ (Line 7).

Algorithm 1 Libgcrypt's Double-and-Always-Add Multiplication.

Input: A positive scalar $k = k_{n-1} \cdots k_0$ and an elliptic-curve point G.
Output: $[k]G$.

```
1: procedure DOUBLE-ALWAYS-ADD(k, G)
2:     Q ← O                                  ▷ O is identity element
3:     t ← truncate-leading-zeros(k)
4:     for i ← t − 1 to 0 do
5:         Q ← 2Q                             ▷ always double Q
6:         T ← Q + G
7:         Q, T ← ct-swap(Q, T, k_i)          ▷ swap if k_i = 1
8:     return Q
```

A side-channel vulnerability. As observed by Minerva [42], Libgcrypt 1.8.4 optimizes the computation time of the main loop (Line 4) of Algorithm 1 by truncating the leading zeros, in case such are present (Algorithm 1, Line 3). Thus, by attempting to detect the number of loop iterations, we are able to determine the amount of leading zeros in the nonce k, allowing us to extract the target's signing key using the lattice reduction techniques discussed in Appendix A.

5.2 Observing ECDSA Leakage

We now describe our approach for estimating the number of loop iterations in Line 3 of Algorithm 1 using audio-interface leakage signal that was conveyed via a VoIP call.

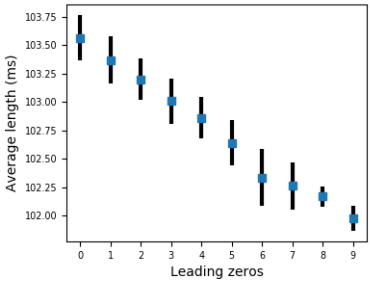


Figure 11: Running time of Libgcrypt's signing operation for different nonce lengths.

Measuring true signing times. First, we ascertain the existence and quality of the conjectured timing variability. To this end, we compiled Libgcrypt 1.8.4, and measured the execution time of its 521-bit ECDSA signing operation on the Acer S3 laptop from Section 4 using cycle accurate timers (i.e., `rdtsc`). For stability and reproducibility, we clamped the CPU at its lowest frequency setting (i.e., highest power saving mode). This avoids disruptive effects such as thermal throttling resulting from long experimental runs with high CPU use, and allowed us to obtain consistent benchmarks.

Figure 11 shows the running time of 25,000 signing operations, as a function of number of leading zeros in the ECDSA nonce. The running time of Libgcrypt's signing operation is linearly inverstly proportional to the number of leading zeros. This leakage stems in the side-channel vulnerability in Line 3 of Algorithm 1, where the number of iterations of the main loop is directly influenced by the number of leading zeros in the signature's nonce.

Observing the leakage remotely over VoIP. We used the Acer S3 laptop, and a Mumble session between the target and attacker's machine as in Section 4.1. While the attacker was recording the call on their end, we made the Acer laptop victim repeatedly execute Libgcrypt ECDSA signing operations.

Figure 12a shows the signal acquired by this attacker. The ECDSA signing operation manifested as a clear gap in the visualized signal (marked in red), lasting about 100 msec. Thus, the timing of the ECDSA computation is observable on the attacker's side of the VoIP call.

5.3 ECDSA Key Recovery

Having established the ability to observe ECDSA timings through VoIP sessions, we proceed to demonstrate the recovery of cryptographic secrets from the observed data.

Experimental setup. Emulating a generic ECDSA target, we wrote a small wrapper application that listens on a network connection and receives a message to sign. The wrapper then uses Libgcrypt 1.8.4 to sign the received message using a fixed signing key generated beforehand, and sends the resulting signature to the attacker.

In parallel to our Libgcrypt wrapper, we also initiated a Mumble session between the Acer laptop and the attacker's machine, recording the audio signal corresponding to each signature operation as received at the attacker's side. Finally, to emulate a realistic networking environment where the attacker and target machines are far apart, we routed the traffic between the target and the attacker machine via a transatlantic VPN connection with a latency of 160 msec.

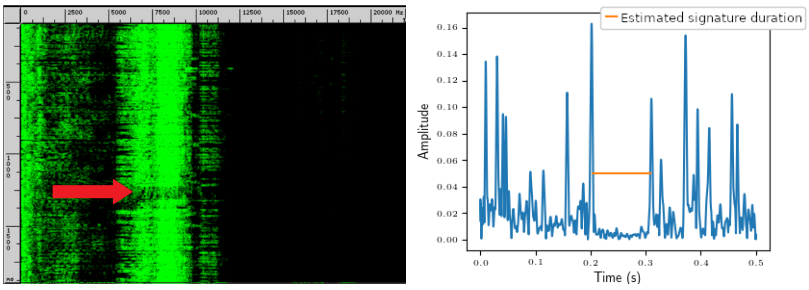
Data collection. We recorded audio at the attacker's side of the conversation, corresponding to 20,000 different messages signed on the Acer target. We signed all 20,000 messages repeatedly in a round-robin fashion until each message was signed 91 times. Since we do not assume accurate synchronization between the attacker and target, we trigger the signatures on the target at roughly regular intervals of 1 sec, used the detector described below in order to locate the signature-dependent leakage in the resulting audio.

Locating the signatures in raw traces. The leakage corresponding to the ECDSA signature manifests as a black region at around 5–10 kHz (see Figure 12a). We thus applied a band-pass filter in the range between 7.4–9.8 kHz to our traces, followed by amplitude demodulation (AM-demod). The resulting signals exhibit distinctly lower values for the duration of the ECDSA-signing operation, as shown in Figure 12b.

Estimating individual signature duration. For each AM-demodulated trace produced as above, we estimated the duration of the signature by scanning the trace for a period of duration of about 100 msec (i.e., one ECDSA signature) where the it drops below a threshold, as visible in Figure 12b. This produces a rough, noisy estimate of the signature's actual duration (Figure 13).

Amplifiying estimation accuracy. As can be seen in Figure 13, many of the traces recorded at the attacker's side are corrupted by noise, presumably generated from unrelated events on the target device. To eliminate these noisy traces, we first grouped the traces by their corresponding message, such that all traces corresponding to the same message belong to the same group. As we target deterministic ECDSA where the nonce is uniquely derived from the key and message, grouping by message guarantees that all the traces belonging to the same group will have the same nonce, and thus the same duration. Within each group, we estimated signature duration as follows: we first remove outliers (signature duration under 80 msec or over 120 msec). We then further cull the remaining traces by dropping those whose estimated duration is farthest from the average, until the standard deviation drops below 0.5 msec. Next, we compute the average estimated duration of the group's remaining traces. Note that the ability to amplify accuracy via processing multiple traces of the same message (and thus the same nonce) stems from RFC 6979's determinism, originally introduced as an attack mitigation.

Key extraction. Each group corresponds to a specific message and its deterministically-derived nonce, which has a specific number of leading zeros. To recover the target's singing key, we chose the groups whose estimated duration is shortest, with the hope that these correspond to nonces with at least



(a) Spectrogram (b) AM demodulation

Figure 12: (Left) Spectrogram corresponding to the ECDSA leakage signal, as recorded on the attacker's side through a Mumble VoIP session. (Right) Trace of an ECDSA signature after bandpass filtering and AM demodulation; the orange line indicates the actual signature duration as measured on the victim device.

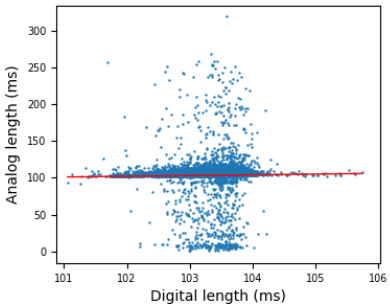


Figure 13: Signature actual duration (x-axis) and our estimation (y-axis); in red: linear regression with incline 1.

leading 6 zeros.

Specifically, we chose the 115 groups whose duration estimates are the shortest, which resulted in only 4 groups out of the 115 having 5 (less than 6) leading zeros.[2] From these 115 groups, we repeatedly sampled a uniformly-drawn subset of 103 groups and provided it as input to the `bkz-enum` solver outlined in Appendix A. Running these in parallel on an Amazon EC2 *c5.9xlarge* 36-threads instance, the correct key was extracted within 20 minutes, at a total computational cost of less than $1.[3]

Parametrization. An important attack parameter is the number of traces acquired. The required number of groups (i.e., distinct signatures) is implied by the underlying lattice-based algorithm; but the *number of traces per group* depends on the leakage and our ability to analyze it to identify signatures with 5 leading zeros. In the above attack, we used 91 traces per signature. Figure 14 shows the effect of varying this parameter: if group size is significantly decreased, an excessive number of false positives occur, i.e., groups are incorrectly detected as having 5 leading zeros, resulting in failure of the lattice-based algorithm.

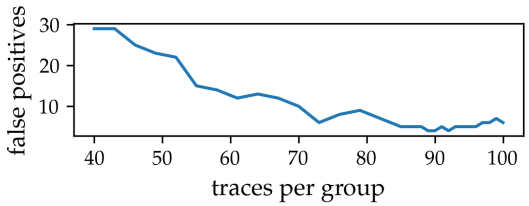


Figure 14: Number of false-positive signatures among the 115 whose duration estimates were shortest, as a function of the number of traces per group (used to estimate the duration).

6 Leakage in Multiplayer Games: the Killer Application

We investigate another attack scenario: online multiplayer games, where players are incentivized to learn the hidden locations and viewpoints of their opponents. We show how a player can use an audio signal from a VoIP call with another player to detect an ambush, giving them a significant advantage. Unlike other game exploits, this is a purely passive attack which does not require modifying the game's code or its runtime environment, and thus cannot be mitigated by standard game anti-cheat measures.

Counter-Strike (CS). CS is one of the best-known and commercially-successful online first-person-shooter games (e.g., over 2019 it maintained an average of over 10,000 concurrent online players on its official gaming hub *Steam*. CS is also a textbook example of an *e-sport*, as a game competitively in an orchestrated league involving millions of dollars worth of prizes and millions of viewers watching livestreamed games. The entire e-sport industry has surpassed $10B in value , and much like in traditional sports, e-sport games face threats of misconduct for players' performance enhancement, and these prominently include exploitation technical vulnerabilities in the game software, i.e., "cheats".

CS: technical background. In CS, every game session has a 3D virtual environment (*map*), and each player controls a person-like avatar located on it, whose purpose is to "kill" other avatars. CS simulates a close-quarters combat scenario where players can gain tactical advantage by using the map's layout, such as using inanimate objects (like walls and cars) for taking shelter and laying ambush.

CS works in a distributed client-server model: anyone can run a server node, and publish it on-line for other players to join. To play against each other, players must connect to the

[2] We learned this information in hindsight, using the ground truth and after our attack was complete.

[3] When performing a longer trial, we saw the solver succeed in extracting the key for 28 out of 360 random subset.

same server. The game state is synchronized between the server and clients as follows: each client periodically sends its avatar's location and viewing angle to the server. Clients also get updates from the server about all existing objects in the avatar's *frustum*, i.e., their field of vision, which is a cone-like area on the map. The player's view is then rendered locally on the client, using this information.

Often, objects are occluded by other objects in the avatar's view, and are thus invisible to the player. However, as long as an object is in the frustum and regardless of occlusion, the client receives information about it and includes it in its rendering process; the occlusion computation is handled within the client's local rendering pipeline. The client's local processing is thus affected by all objects within the frustum.

It is common for CS players to maintain an open audio channel with other players on the map via VoIP software, a popular choice of which is Mumble.

Frustum content leaks to opponents via audio. It follows from the above that each player's physical processor behavior is affected by the objects within that player's frustum, like other players' avatars, whether or not those objects occluded. Thus, audio recordings from a client's built-in microphone, transmitted over VoIP, might leak information about their frustum's content.

A simple attack: camper detection. In this scenario, the attacker cheats by detecting whether an opponent is *camping*, i.e. taking shelter and hiding in ambush, in a particular spot (note that CS maps typically have well-known spots that are especially suitable for camping. *If* the opponent is camping in the suspected spot, then their likely frustum is known to the attacker. The attacker can attempt to detect the ambush by moving its avatar to enter and exit the camper's supposed frustum in a place that the enemy does not have a direct line of sight to (for example, behind a wall), and examining the audio signal from their environment. Whenever the attacker enters/exits the frustum, the camper's processors starts/stops including the attacker's avatar in the rendering pipeline. The resulting effect on the camper's processor is detectable by the attacker, by observing the spectrum of the audio signal transmitted by the camper over Mumble. Figure 15 illustrates this. *Otherwise*, if the attacker is not camping in the suspected location, it is unlikely that the audio signal will be correlated with the attacker's movements.

CS attack scenario. The attack operates in a scenario where CS opponents have a VoIP channel between them (as is common in CS "player versus player" games). No physical proximity is assumed. This attack is applicable when the target player is carrying out the "camper" strategy, i.e., stays mostly still and fixes their gunsight on a point where their opponent (the attacker) is expected to emerge.

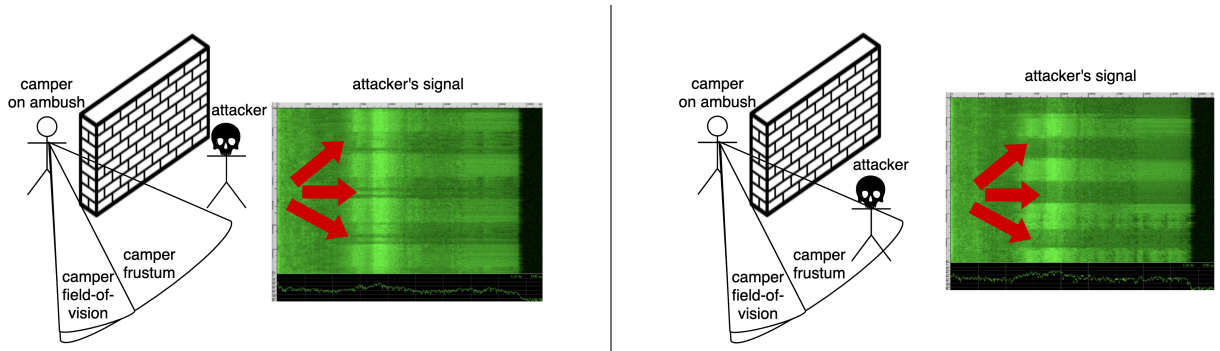
Simulating the attack in a CS battle.[4] For this experiment (illustrated by the aforementioned Figure 15, we simulated a "camper" player who plays CS on an Acer S3 laptop that runs Ubuntu 20.04. Our camper and attacker play a one-on-one game of Counter-Strike on a third-party server (hosted on the Internet) with VAC turned on. They also keep an open Mumble audio channel.To force the Mumble connection to traverse the Internet (even though both machines were physically in our lab), we connected both machines to a cloud hosted VPN server by OpenVPN.

These players joined one of CS's standard gaming arenas named *cs-assault* which features a truck with a cargo container. Our attacker is inside the cargo container, and the camper could be lurking on *either* side of the truck. In this classic scenario, the attacker would normally not know which side of the track to face when emerging from the container; if they guess wrong, they will be an easy target as the camper is ready to shoot them right after their next step. To discover the camper's location, our attacker repeatedly moves to the right and to the left, and detects which of the movement causes a change in the VoIP audio sent by the camper, which indicates entering into their frustum. See Figure 16. Our attacker successfully detects the camper's ambush on the right, and thus decided to exit the truck and go left, circumvent the truck all the way to the right side and "kill" their opponent's avatar from behind. We refer the interested reader to the **video recording** of this demo[5]. We have also tested other maps and camping situations with different geometries, with similar results.

The attack circumvents standard mitigations. A well-known cheating technique in multiplayer games is manipulation of client code. For example, the CS client can be modified to directly extract and display other players' avatars, regardless of occlusions. This requires either modifying the game's executables, installing an extension plugin, or attaching a debugger to the game process. Such attacks can sometimes be mitigated by attestation mechanisms such as SGX [9]. The game's developer, Valve, has integrated into it a mechanism called Valve Anti Cheat (VAC), which detects cheats based on process memory signatures and anomalies. VAC-enabled game servers send periodic challenges to game clients, which must then answer the challenge by revealing relevant information about the process memory (or risk raising an anomaly). Cheating players are permanently banned from playing in such servers [71]. Cheat developers adapt to VAC's signatures by modifying their code, resulting in a repetitive cat-and-mouse dynamic. Our attack circumvents VAC entirely, since it does not require any modification to the game environment: nothing runs on any of the game client computers other than the unmodified game and a VoIP application. The signal analysis does not have any direct interaction with the game process, and can be executed on a separate machine.

[4] We have disclosed the details of this attack to the game's developer.

[5] https://vimeo.com/468851232



(a) Attacker is outside the frustum

(b) Attacker is inside the frustum

Figure 15: Counter-Strike camper detection via side channel leakage: when the attacker is within the camper's frustum, rendering on the camper's computer results in visible differences in the spectrogram. Thus, by entering the frustum and examining the signal, the attacker can detect the ambush, while neither seeing nor being seen by the camper.

7 Countermeasures

Signal-generating hardware. As discussed in Section 2.3, the leakage we observed originates in electromagnetic emanations. These can be attenuated at the hardware level by better shielding of radiation and filtering of currents and voltages; though this is difficult and expensive to achieve at the level of power-hungry components like CPUs. It is possible that in some other computers, acoustic leakage occurs as well (e.g., of the effects reported in [29]), in which case acoustic blocking or baffling would also be required, raising costs and posing challenges to air flow for cooling.

Sensors. Another approach is to protect the sensors and associated circuitry from leakage. For example, noise can be picked up by microphone's amplifier and ADC, due to EM emanations impinging on their ground and power connections. By shielding and filtering of these connections, signals can be attenuated. As these are low-power components, this can be done relatively inexpensively (as done in high-fidelity audio interfaces), and moreover provide the side benefit of improving signal quality also for normal audio recording use.

Software. Standard techniques for writing leakage-resilient software are applicable: most crucially constant-time algorithms for processing sensitive data [11, 43], but also finer-granularity DPA protections (e.g., masking) [20, 53].

Signal processing. Even if leakage-bearing signals have been acquired and digitized by the sensor, the undesired signal can be intentionally degraded. Crude methods include adding sufficient digital noise to reduce signal-to-noise ratio below usefulness for the attacker, or low-pass filtering of the signal to remove the high frequencies where the leakage is most salient. We pose the open question of finding good tradeoffs between attenuation (or other disruption) of the undesirable leakage signal vs. preservation of the desirable signals (e.g., under psychoacoustic measures).

Data sharing. Lastly, in the absence of adequate mitigations at the aforementioned levels, one should assume that sensor recordings are no less sensitive than any computation (or computed-upon data) that occurred locally during the recording. Their distribution should be limited accordingly. This would also mitigate related attacks on peripherals, such as acoustically-captured keystrokes [6] or screen content [27].

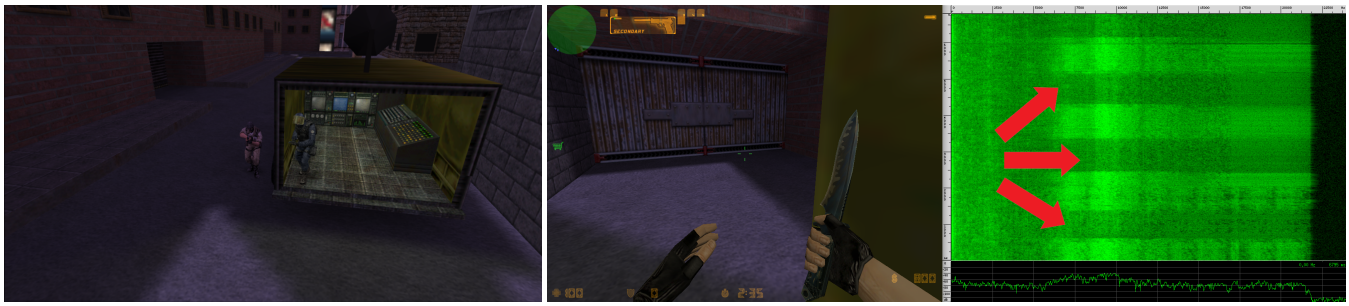
8 Limitations and Future Work

Additional sensors and devices. We pose an open question the exploration of computational leakage captured by other common sensors. For example, webcam sensors contain an array of sensitive analog-to-digital converters, which can be affected by EM emanations, and the resulting videos are often shared as recordings or video calls. Likewise, investigating the presence of similar leakage on other device categories, such as phones and tablets (which have even more sensors), is an important open question.

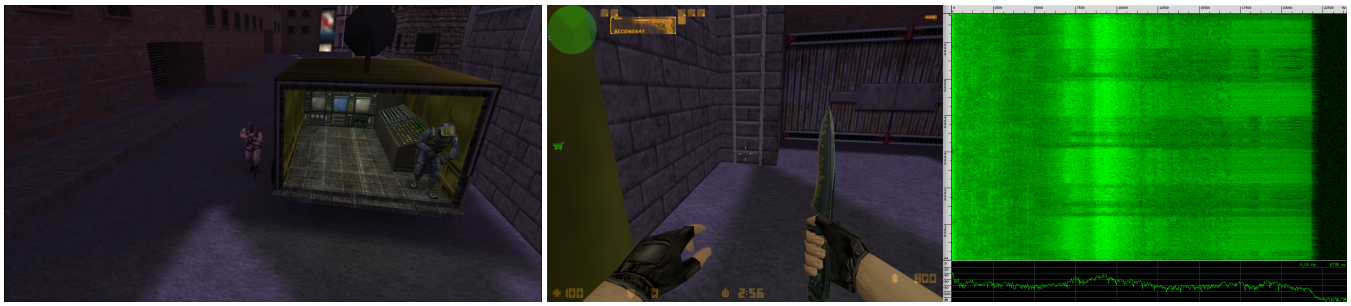
Attacking additional targets. The software targets considered in this work are not adequately hardened against physical side channel attacks, and appear to not have been designed with such threat model in mind. Thus, our work should be viewed as a feasibility result, establishing the possibility of attacking PCs using internal analog sensors. We leave the task of attacking side channel harden primitives (e.g., AES-NI) to future work. Finally, our work side steps the low sampling rate provided by internal audio interfaces, by heavily relying on the attacker's ability to average multiple traces corresponding to the same leakage. While this is possible in some cases (e.g., website distinguishing or deterministic-ECDSA), we leave the task of attacking randomized primitives, such as regular (EC)DSA or (EC)DH, to future work.

Cross-device training/testing. Across our experiments we used the same device for collecting training and testing sets. This simulates a scenario where the attacker has access to the victim device for collecting traces. We expect our trained models to generalize across different physical devices, yet achieving this would require training on a cluster of multiple

(a) Attacker's avatar (back of truck) is outside the frustum of the victim (outside the truck).



(b) Attacker's avatar stepped to the front of the truck, entering the victim's frustum. This causes a noticeable change in the signal.



(c) Attacker's avatar is again out of the victim's frustum. The signal is similar to the first case.

Figure 16: Counter-Strike attack: a side-channel attacker (avatar inside truck) detects an ambush by a camper (outside truck). Left: spectator's view. Center: attacker's view. Right: signal acquired by attacker.

devices of the same model [27] (the attacker must still know the model). Operating on a limited budget, we preferred allocating it for acquiring a diverse set of devices, aiming to show the prevalence of the leakage, rather than focusing on specific device models. We acknowledge this limitation, leave this task to future work.

Practicability of website distinguishing. While our results in Section 4 show that the leakage signal can be used to differentiate between websites even in the presence of ample ambient noise, there remain important limitations a real-world attacker would have to address. First, the attacker's accuracy varies depending on the victim's device and the set of websites being distinguished (see Table 2). Second, our experiments mostly model a system where each website visit is performed under fairly consistent conditions, i.e., without having other websites open and while only having the OS's default background activities running. While we observe that spectra do remain distinguishable even in the presence of extraneous activity (Figure 10), we leave the task of addressing these limitations to future work. Third, we assume the attacker can record for at least 40 seconds while the website is in the foreground, and that the user performs no activity such as scrolling or switching tabs during the recording. While the attacker might try to detect and discard audio traces that include user activity via the recorded audio stream (e.g., mouse/keyboard clicks), we leave this issue as a future research question. Finally, the demonstrated attack assumes that the attacker obtains a training dataset which corresponds to the contents of the websites actually rendered by the victim. In practice however, websites' contents often changes over time, geographical location, or user configuration (e.g., dark theme and ad blocking). All of these affect the leakage signal, reducing similarity with the

leakage profiled by the attacker, thereby reducing the attack's accuracy. We leave the characterization and mitigation of these effects to future efforts.

Acknowledgments

This work was supported by the Air Force Office of Scientific Research (AFOSR) under award number FA9550-20-1-0425; by the Blavatnik Interdisciplinary Cyber Research Center (ICRC); by the Check Point Institute for Information Security; by Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL) under contracts FA8750-19-C-0531 and HR001120C0087; by the Israeli Ministry of Science and Technology; by the National Science Foundation under grant CNS-1954712; and gifts from Intel, AMD, and VMware. Any opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect the views of AFOSR, AFRL, NSF, the U.S. Government or other sponsors.

References

[1] Acıiçmez, O., Koç, Ç.K., Seifert, J.P.: Predicting secret keys via branch prediction. In: CT-RSA 2007

[2] Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side-channel(s). In: CHES 2002

[3] Albrecht, M.R., Heninger, N.: On bounded distance decoding with predicate: Breaking the "lattice barrier" for the hidden number problem. IACR Cryptol. ePrint Arch. (2020)

[4] Aldaya, A.C., Brumley, B.B., ul Hassan, S., García, C.P., Tuveri, N.: Port contention for fun and profit. In: IEEE S&P Symposium 2019

[5] Andrysco, M., Kohlbrenner, D., Mowery, K., Jhala, R., Lerner, S., Shacham, H.: On subnormal floating point and abnormal timing. In: IEEE S&P Symposium 2015

[6] Asonov, D., Agrawal, R.: Keyboard acoustic emanations. In: IEEE S&P Symposium 2004

[7] Backes, M., Dürmuth, M., Gerling, S., Pinkal, M., Sporleder, C.: Acoustic side-channel attacks on printers. In: USENIX Security 2010

[8] Balzarotti, D., Cova, M., Vigna, G.: Clearshot: Eavesdropping on keyboard input from video. In: IEEE S&P Symposium 2008

[9] Bauman, E., Lin, Z.: A case for protecting computer games with sgx. In: SysTEX 2016

[10] Berger, Y., Wool, A., Yeredor, A.: Dictionary attacks using keyboard acoustic emanations. In: CCS 2006

[11] Bernstein, D.J., Lange, T., Schwabe, P.: The security impact of a new cryptographic library. In: LATINCRYPT 2012

[12] Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in diffie-hellman and related schemes. In: CRYPTO 1996

[13] Bos, J.W., Halderman, J.A., Heninger, N., Moore, J., Naehrig, M., Wustrow, E.: Elliptic curve cryptography in practice. In: FC 2014

[14] Breitner, J., Heninger, N.: Biased nonce sense: Lattice attacks against weak ecdsa signatures in cryptocurrencies. In: FC 2019. (2019)

[15] Brengel, M., Rossow, C.: Identifying key leakage of Bitcoin users. In: RAID 2018

[16] Brumley, B.B., Tuveri, N.: Remote timing attacks are still practical. In: ESORICS 2011

[17] Brumley, D., Boneh, D.: Remote timing attacks are practical. Computer Networks (2005)

[18] Camurati, G., Poeplau, S., Muench, M., Hayes, T., Francillon, A.: Screaming channels: When electromagnetic side channels meet radio transceivers. In: CCS 2018

[19] Canella, C., Genkin, D., Giner, L., Gruss, D., Lipp, M., Minkin, M., Moghimi, D., Piessens, F., Schwarz, M., Sunar, B., Van Bulck, J., Yarom, Y.: Fallout: Leaking data on meltdown-resistant cpus. In: CCS 2019

[20] Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: CRYPTO 1999

[21] Clark, S.S., Mustafa, H., Ransford, B., Sorber, J., Fu, K., Xu, W.: Current events: Identifying webpages by tapping the electrical outlet. In: ESORICS 2013

[22] Compagno, A., Conti, M., Lain, D., Tsudik, G.: Don't Skype & type! acoustic eavesdropping in voice-over-ip. In: ASIACCS 2017

[23] Courtois, N.T., Valsorda, F., Emirdag, P.: Private Key Recovery Combination Attacks: On Extreme Fragility of Popular Bitcoin Key Management, Wallet and Cold Storage Solutions in Presence of Poor RNG Events. (2014)

[24] Enev, M., Gupta, S., Kohno, T., Patel, S.N.: Televisions, video privacy, and powerline electromagnetic interference. In: CCS 2011

[25] Evtyushkin, D., Riley, R., Abu-Ghazaleh, N.C., ECE, Ponomarev, D.: Branchscope: A new side-channel attack on directional branch predictor. ACM SIGPLAN Notices (2018)

[26] Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: CHES 2001

[27] Genkin, D., Pattani, M., Schuster, R., Tromer, E.: Synesthesia: Detecting screen content via remote acoustic side channels. In: IEEE S&P Symposium 2019

[28] Genkin, D., Pipman, I., Tromer, E.: Get your hands off my laptop: Physical side-channel key-extraction attacks on pcs. Journal of Cryptographic Engineering (2015)

[29] Genkin, D., Shamir, A., Tromer, E.: RSA key extraction via low-bandwidth acoustic cryptanalysis. In: CRYPTO 2014

[30] Giechaskiel, I., Eguro, K., Rasmussen, K.B.: Leakier wires: Exploiting FPGA long wires for covert-and side-channel attacks. TRETS (2019)

[31] Giechaskiel, I., Rasmussen, K., Szefer, J.: Reading between the dies: Cross-slr covert channels on multi-tenant cloud FPGAs. In: ICCD 2019

[32] Giechaskiel, I., Rasmussen, K.B., Eguro, K.: Leaky wires: Information leakage and covert communication between FPGA long wires. In: ASIACCS 2018

[33] Giechaskiel, I., Rasmussen, K.B., Szefer, J.: Measuring long wire leakage with ring oscillators in cloud FPGAs. In: FPL 2019

[34] Gnad, D.R., Krautter, J., Tahoori, M.B.: Leaky noise: new side-channel attack vectors in mixed-signal iot devices. CHES 2019

[35] Gnad, D.R., Nguyen, C.D.K., Gillani, S.H., Tahoori, M.B.: Voltage-based covert channels in multi-tenant FPGAs. IACR Cryptol. ePrint Arch. (2019)

[36] Gravellier, J., Dutertre, J.M., Teglia, Y., Moundi, P.L., Olivier, F.: Remote side-channel attacks on heterogeneous soc. In: CARDIS. (2019)

[37] Großschädl, J., Oswald, E., Page, D., Tunstall, M.: Side-channel analysis of cryptographic software via early-terminating multiplications. In: ICISC 2009

[38] Halevi, T., Saxena, N.: A closer look at keyboard acoustic emanations: random passwords, typing styles and decoding techniques. In: CCS 2012

[39] Halevi, T., Saxena, N.: Keyboard acoustic side channel attacks: exploring realistic and security-sensitive scenarios. International Journal of Information Security (2015)

[40] Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your ps and qs: Detection of widespread weak keys in network devices. In: USENIX Security 2012

[41] Hunter, J.D.: Matplotlib: A 2D graphics environment. Computing in science & engineering (2007)

[42] Jancar, J., Sedlacek, V., Svenda, P., Sys, M.: Minerva: The curse of ECDSA nonces systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces. CHES 2020

[43] Käsper, E., Schwabe, P.: Faster and timing-attack resistant AES-GCM. In: CHES 2009

[44] Kenjar, Z., Frassetto, T., Gens, D., Franz, M., Sadeghi, A.R.: V0ltpwn: Attacking x86 processor integrity from software. In: USENIX Security 2020

[45] Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., et al.: Spectre attacks: Exploiting speculative execution. In: IEEE S&P Symposium 2019

[46] Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: CRYPTO 1999

[47] Kocher, P., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. Journal of Cryptographic Engineering (2011)

[48] Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: CRYPTO 1996

[49] Kuhn, M.G.: Compromising emanations: eavesdropping risks of computer displays. PhD thesis, University of Cambridge (2003) Technical Report UCAM-CL-TR-577, PhD dissertation.

[50] Lipp, M., Kogler, A., Oswald, D., Schwarz, M., Easdon, C., Canella, C., Gruss, D.: PLATYPUS: Software-based Power Side-Channel Attacks on x86. In: IEEE S&P Symposium 2021

[51] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M.: Meltdown: Reading kernel memory from user space. In: 27th USENIX Security Symposium (USENIX Security 18)

[52] Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards. Springer Science & Business Media (2007)

[53] Messerges, T.S.: Securing the AES finalists against power analysis attacks. In: FSE. (2000)

[54] Moghimi, D., Sunar, B., Eisenbarth, T., Heninger, N.: Tpm-fail:TPM meets timing and lattice attacks. In: USENIX Security 2020

[55] Murdock, K., Oswald, D., Garcia, F.D., Van Bulck, J., Gruss, D., Piessens, F.: Plundervolt: Software-based fault injection attacks against intel sgx. In: IEEE S&P Symposium 2020

[56] Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of aes. In: CT-RSA 2006

[57] O'Flynn, C., Dewar, A.: On-device power analysis across hardware security domains. CHES 2019

[58] Percival, C.: Cache missing for fun and profit. Presented at BSDCan. http://www.daemonology.net/hyperthreading-considered-harmful (2005)

[59] Pornin, T.: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA), RFC 6979. https://tools.ietf.org/html/rfc6979 (August 2013)

[60] Provelengios, G., Holcomb, D., Tessier, R.: Characterizing power distribution attacks in multi-user FPGA environments. In: FPL 2019

[61] Provelengios, G., Ramesh, C., Patil, S.B., Eguro, K., Tessier, R., Holcomb, D.: Characterization of long wire data leakage in deep submicron FPGAs. In: FPGA Conference 2019

[62] Qiu, P., Wang, D., Lyu, Y., Qu, G.: Voltjockey: Breaching trustzone by software-controlled voltage manipulation over multi-core frequencies. In Cavallaro, L., Kinder, J., Wang, X., Katz, J., eds.: CCS 2019

[63] Quisquater, J.J., Samyde, D.: Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In: E-smart 2001. (2001)

[64] Ramesh, C., Patil, S.B., Dhanuskodi, S.N., Provelengios, G., Pillement, S., Holcomb, D., Tessier, R.: FPGA side channel attacks without physical access. In: FCCM 2018

[65] Rasmussen, K., Giechaskiel, I., Szefer, J.: Capsule: Cross-FPGA covert-channel attacks through power supply unit leakage. In: IEEE S&P Symposium 2020

[66] Schellenberg, F., Gnad, D.R., Moradi, A., Tahoori, M.B.: An inside job: Remote power analysis attacks on FPGAs. In: DATE 2018

[67] Schellenberg, F., Gnad, D.R., Moradi, A., Tahoori, M.B.: Remote inter-chip power analysis side-channel attacks at board-level. In: ICCAD 2018

[68] Schuster, R., Shmatikov, V., Tromer, E.: Beauty and the burst: Remote identification of encrypted video streams. In: USENIX Security 2017

[69] Tang, A., Sethumadhavan, S., Stolfo, S.: CLKSCREW: Exposing the perils of security-oblivious energy management. In: USENIX Security 2017

[70] Valin, J., Maxwell, G., Terriberry, T.B., Vos, K.: High-Quality, Low-Delay Music Coding in the Opus Codec. (2016)

[71] Valve: Valve anti-cheat system. https://support.steampowered.com/kb/7849-RADZ-6869/ accessed: August 2020.

[72] Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In: USENIX Security 2018. (2018) 991–1008

[73] Van der Walt, S., Schönberger, J.L., Nunez-Iglesias, J., Boulogne, F., Warner, J.D., Yager, N., Gouillart, E., Yu, T.: scikit-image: image processing in python. PeerJ (2014)

[74] Van Eck, W.: Electromagnetic radiation from video display units: An eavesdropping risk? Computers & Security (1985)

[75] van Schaik, S., Milburn, A., Österlund, S., Frigo, P., Maisuradze, G., Razavi, K., Bos, H., Giuffrida, C.: RIDL: Rogue in-flight data load. In: IEEE S&P Symposium 2019

[76] van Schaik, S., Minkin, M., Kwong, A., Genkin, D., Yarom, Y.: Cacheout: Leaking data on intel cpus via cache evictions (2020)

[77] Vuagnoux, M., Pasini, S.: Compromising Electromagnetic Emanations of Wired and Wireless Keyboards. In: USENIX Security 2009

[78] Yarom, Y., Falkner, K.: FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In: USENIX Security 2014

[79] Zhao, M., Suh, G.E.: FPGA-based remote power side-channel attacks. In: IEEE S&P Symposium 2018

[80] Zhuang, L., Zhou, F., Tygar, J.D.: Keyboard acoustic emanations revisited. TISSEC 2009

A Using Partial Information to Extract Keys

We employ the key-recovery approach of Minerva [42] (itself based on [12]) with the recent improvements of [3]. In the following, we briefly review the algorithmic approach.

Suppose we obtain a set of signatures each of which has a *short nonce*, i.e., k that has many leading zeros. The nonce

are secrets, but we will be able to identify the signing operations that involve short nonces using the timing variability discussed above, observed through the side channel. Given such a set of signatures, extracting the ECDSA secret key is reduced to a Hidden Number Problem:

HNP. The *Hidden Number Problem (HNP)* [12] is: find a secret $\alpha \in \mathbb{Z}_p$ given a set of integer pairs $\{(t_i, u_i)\}_i$ and leakage parameter l such that $|\alpha t_i - u_i|_p < p/2^l$ for each i. Here, $|\cdot|_p$ means reducing modulo p into the range $[0, p-1]$.

From ECDSA leakage to HNP [12]. Suppose we are given a set of m signed messages. For each, we know the message M with hash digest z and the signature (r,s). Moreover, suppose that through side-channel observations, we learned that all of these signatures have a *short nonce*: the l leading bits of k are zero (for some small fixed l to be chosen later). Rearranging the formula of s from Section 5.1 we get $rs^{-1}d + zs^{-1} = k \mod n$. From the leakage information we know that $k < n/2^l$. Letting $t = rs^{-1}$ and $u = -zs^{-1}$, we achieve the inequality $|t \cdot d - u|_n = k < n/2^l$ as in the definition of HNP. We take the set (u_i, t_i) as our input to a HNP problem with the secret key d as the hidden number.

At this point we diverge from [12,42] and use the improved approach described and implemented in [3], which can practically succeed using a smaller set signatures.

Lattices. Given a matrix B in $\mathbb{R}^d$ with rows $b_1, ..., b_d$ we define the lattice spanned by B to be the discrete subgroup: $\Lambda(B) = \{v_i b_i | v_i \in \mathbb{Z}\}$

SVP. The *Shortest Vector Problem (SVP) with predicate ($uSVP_{f(\cdot)}$)* [3] is: given a lattice Λ and a predicate f on vectors, find the shortest nonzero vector $v \in \Lambda$ such that $f(v) = 1$. This is a computationally difficult problem in the general case, but [3] offers heuristic algorithms that are practical in our parameter setting.

From HNP to uSVP with predicate [3]. Consider the lattice generated by the following base:

$$\begin{pmatrix} n & 0 & & 0 & 0 & 0 \\ 0 & n & \cdots & 0 & 0 & 0 \\ & \vdots & \ddots & \vdots & & \\ 0 & 0 & \cdots & n & & \\ t_1 & t_2 & \cdots & t_m & 1/2^l & 0 \\ u_1 & u_2 & \cdots & u_m & 0 & n/2^l \end{pmatrix}$$

This lattice contains the point $v = (k_1, k_2, ..., k_m, d/2^l, -n/2^l)$ which can by obtained by multiplying the second to last basis vector by the secret key, subtracting the last vector and adding other rows for the correct modulus. This vector has the Euclidean norm that is less than $\sqrt{m+2} \cdot n/2^l$. Given this vector, the secret key can be extracted from the second to last entry. We use this lattice basis as input to the solver. For the predicate we take a function that checks whether a vector reveals the secret key. Note that $(0, 0, ..., 0, 2^l, 0)$ is also a point in the lattice and shorter than v, so using a simple SVP solver might not give us the desired result. This lattice is then improved in two ways, described in the following.

Assuming a larger leakage. The solver's success probability can be improved in a way that is equivalent to learning one extra bit for each nonce. Notice that our goal is to construct a lattice such that the values k_i are small in absolute value to create a short vector, but we have $0 \le k_i \le 2^l$ and $t_i \cdot d - u_i - \chi_i \cdot n = k$ for each i and some $\chi_i \in \mathbb{Z}$. This means:

$$|t \cdot d - u - n/2^{l+1} - \chi_i \cdot n| = |k - n/2^{l+1}| < n/2^{l+1}$$

We set $u'_i = u_i + n/2^{l+1}$ and $k'_i = k_i - n/2^{l+1}$ which gives us a lattice basis with a point with norm less than $\sqrt{m+2} \cdot n/2^{l+1}$, which is shorter and thus easier to find by the solver.

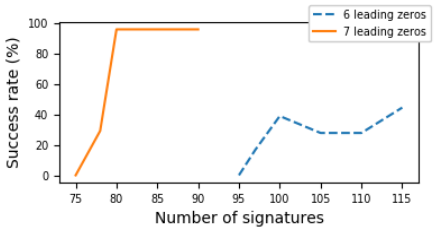


Figure 17: Performance of the uSVP with predicate solver, for various number of signatures and number of leading zeros.

Eliminating the secret key. This optimization gives us a reduction from our previous lattice into a lattice of dimension smaller by 1: notice we have m equations of the form $t_i \cdot d - u'_i = k'_i \mod n$ for each $1 \le i \le m$. Isolating d from the equations allows us to compare between different equations. Choosing the last one, we get that for every $1 \le i \le m-1$ we have $t_i^{-1} \cdot (u_i + k_i) = t_m^{-1} \cdot (u_m + k_m) \mod n$. Rearranging yields: $u'_i - t_i \cdot t_m^{-1} \cdot u'_m + k_i = t_i \cdot t_m^{-1} \cdot k_m \mod n$. We set $u'' = u' - t_i \cdot t_m^{-1} \cdot u'_m$ and $t'' = t_i \cdot t_m^{-1}$ and we get a new HNP instance with $m-1$ equations and k'_m as the hidden number. We now create a lattice basis as before:

$$\begin{pmatrix} n & 0 & & 0 & 0 & 0 \\ 0 & n & \cdots & 0 & 0 & 0 \\ & \vdots & \ddots & \vdots & & \\ 0 & 0 & \cdots & n & & \\ t''_1 & t''_2 & \cdots & t''_m & 1 & 0 \\ u''_1 & u''_2 & \cdots & u''_m & 0 & n/2^{l+1} \end{pmatrix}$$

This lattice contains the point $(k'_1, k'_2, ..., k'_m, -n/2^l)$ which is obtained by multiplying the second to last basis vector to k''_m and subtracting the last vector. This vector has norm of less than $\sqrt{m+1} \cdot n/2^{l+1}$. This lattice, with the same predicate, is then fed to the aforementioned solver [3] to extract the key.

Solver performance. We ran the solver on generated data to estimate its performance under different set of parameters. Figure 17 shows the success rate on the values that were tested. We chose 103 signatures with 6 leading zeros or more as our target amount of information needed to accumulate in order to extract the secret key.