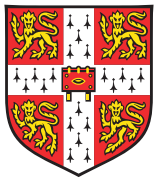
Eavesdropping risks of the DisplayPort video interface

Dimitrije Erdeljan



University of Cambridge
Department of Computer Science and Technology
King's College

October 2023

This dissertation is submitted for
the degree of Doctor of Philosophy

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text.

This dissertation does not exceed the regulation length of 60 000 words, including tables and footnotes.

Eavesdropping risks of the DisplayPort video interface

Dimitrije Erdeljan

Summary

The switching activity of digital circuits unintentionally generates electromagnetic signals, which may leak sensitive information processed by the device to nearby radio receivers. This problem, known as *compromising emanations* or *TEMPEST*, has been demonstrated for computer displays using analog video interfaces (VGA) and older digital interfaces (LVDS, HDMI, DVI). DisplayPort is a newer interface with a significantly more complex signal structure, and in particular uses a linear-feedback shift register to scramble the transmitted pixel data. Due to scrambling, images produced by applying previously published eavesdropping techniques to DisplayPort appear as random noise, and the interface is thought to be a far more difficult target.

I start by showing that DisplayPort is vulnerable to electromagnetic eavesdropping, assuming that the displayed image mainly consists of a small set of colours. The attack starts by recovering scrambler timing parameters and synthesising a replica of the scrambler synchronised with the target. This replica is then used to build templates for each of the expected colours, and to identify pixel colours from short-term cross-correlation between the received signal and templates.

The two main limitations of this initial attack are limited accuracy of the reset-timing model and a requirement that the attacker already knows which colours are present in the image. I address the former by designing a scrambler tracking algorithm based on a phase-locked loop that keeps the local replica closely synchronised with the target. For the latter, I exploit several properties of the 8b/10b encoding used together with this accurate scrambler alignment to efficiently enumerate colours and produce a list of candidate colours likely to be present in the image.

Finally, I extend the tracking algorithm to also align signal phase across frames, which enables coherent periodic averaging of template correlations. This averaging technique further improves the signal-to-noise ratio in the reconstructed image and thus increases eavesdropping range. Accurate time alignment additionally improves horizontal resolution over that achieved using the simpler timing model. I demonstrate that the algorithms developed in this thesis can be used to recover clearly readable text from 8 m distance in realistic circumstances, even using a software-defined radio receiver with a bandwidth that is an order of magnitude lower than the bitrate used in the DisplayPort video link.

Acknowledgments

First of all, I am grateful to my supervisor, Markus Kuhn, for his guidance, advice, and many technical discussions, without which this thesis and the ideas in it would never have happened.

Thanks also to the Security Group, CyberSoc, and others for useful discussions and making my time in Cambridge better.

I would also like to thank Zoran Kostić for taking the time to proofread this thesis, and Stefan Velja for both proofreading and formatting advice.

My research was funded by Cambridge Trust and King's College.

Finally, I would like to thank my family and friends for their support. Most of all, I am grateful to Marijana, whose support and encouragement kept me going during these four years.

Contents

Notation

j	Imaginary unit satisfying $\mathrm{j}^2 = -1$
e	Euler's constant $\mathrm{e} = 2.71828\ldots$
$\mathbb{Z}_n$	Set of natural numbers smaller than n: $\mathbb{Z}_n = \left\{m \in \mathbb{Z} \,\middle|\, 0 \leq m < n\right\}$
$a \oplus b$	Bitwise exclusive-or of integers a and b
$\Re\{x\}$	Real part of $x \in \mathbb{C}$
x^*	Complex conjugate of $x \in \mathbb{C}$
$\lvert x \rvert$	Argument of complex number x: $\left\lvert \mu \mathrm{e}^{\mathrm{j}\varphi} \right\rvert = \mu$ for $\mu, \varphi \in \mathbb{R}$
$\angle x$	Phase of complex number x: $\angle\left(\mu \mathrm{e}^{\mathrm{j}\varphi}\right) = \varphi$ for $\mu, \varphi \in \mathbb{R}$, $0 \leq \varphi < 2\pi$
$\lfloor x \rfloor$	Integer part of $x \in \mathbb{R}$: $\lfloor x \rfloor \in \mathbb{Z}$ and $0 \leq x - \lfloor x \rfloor < 1$
$\lceil x \rceil$	$x \in \mathbb{R}$ rounded up: $\lceil x \rceil \in \mathbb{Z}$ and $0 \leq \lceil x \rceil - x < 1$
$\mathbb{E}[X]$	Expected value of random variable X
$\mathbb{V}[X]$	Variance of random variable X: $\mathbb{V}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2]$
$X \sim \mathcal{N}(\mu, \sigma^2)$	Random var. X has Gaussian distribution with mean μ and variance σ^2
$X \sim \mathcal{U}(a, b)$	Random var. X is uniformly distributed over interval $[a, b]$
$X \sim \mathcal{R}(\mu, \sigma)$	Random var. X has Rice distribution with centre μ and scale σ
$\operatorname{sinc} x$	Normalised sinc function: $\operatorname{sinc} x = \frac{\sin \pi x}{\pi x}$
$\mathcal{F}\{h(t)\}(f)$	Fourier transform of a time-domain function $h(t)$ at frequency f
$u * v$	Convolution of discrete-time sequences $u[n]$ and $v[n]$
$\delta(t)$	Dirac delta function
$\operatorname{argmax}_{x \in S} f(x)$	A value $x \in S$ for which $f(x)$ is maximal

Symbols

h_d	Displayed image height in pixels
w_d	Displayed image width in pixels
h_t	Total image height in pixels, including blanking regions
w_t	Total image width in pixels, including blanking regions
f_v	Frame rate
f_h	Line rate $f_\mathrm{h} = h_\mathrm{t} \cdot f_\mathrm{v}$
f_p	Displayed pixel rate $f_\mathrm{p} = w_\mathrm{t} \cdot f_\mathrm{h}$
f_b	DisplayPort bitrate
n_L	Number of active DisplayPort lanes
$\Xi[n]$	DisplayPort scrambling byte sequence
$\xi[n]$	DisplayPort scrambling bit sequence
n_p	Total line length after padding, including the blanking period
n_b	Length of horizontal blanking period after padding
$s[n]$	The received IQ sampled signal
f_s	Sampling rate of the received signal after preprocessing
$z[n]$	A template
$z^+[n], z^-[n]$	A positive-disparity (resp. negative-disparity) template
$\mathbf{X}_\mathrm{o}$	The set of observed scrambler reset times
x_0, R, P	Reset model parameters: initial offset, reset period, and overflow period

Chapter 1

Introduction

The switching activity of digital circuits unintentionally generates tiny electromagnetic pulses. These may not only interfere with radio reception nearby, but can also leak information about processed confidential information. This problem is commonly referred to as *compromising emanations* or *TEMPEST*, especially in government standards that define emission-security limits for hardware as a protection [1].

Among the many radio emissions unintentionally generated by computer circuitry, information leakage from *video signals* can be particularly easy to demonstrate, because these tend to be highly redundant: text is typically displayed with only a small number of colours, each character is represented by many pixels, and the content of these pixels is typically retransmitted 60 times per second (frame refresh rate). Such redundancy helps eavesdroppers to separate emissions from noise.

The readability of the received signals depends on both the video technology used and the type of signal detection applied. The first published electromagnetic eavesdropping demonstrations on video signals targeted cathode-ray tube (CRT) displays, which are driven by *analog video signals*, in which pixel brightness is transmitted as a voltage on a wire (or on three wires, for red, green and blue), one pixel at a time. IBM's VGA connectors were once a common example of this interface type.

In 1985, van Eck [2] used an amplitude modulation (AM) receiver, from a television set, to detect the electromagnetic pulses generated whenever an analog video signal switched the electron beam of a CRT on or off. Such a rising or falling edge can result in a positive impulse at the AM receiver's output. If that is turned into a raster image with the same line and frame rate as those of the eavesdropped display, a text, such as shown in Figure 1.1(a), will appear to the AM eavesdropper as shown in Figure 1.1(b). Vertical strokes appear doubled, because both bright/dark and dark/bright transitions result in an impulse, whereas horizontal strokes appear only as two end points. The receiver's bandwidth should be at least as large as the pixel-clock frequency: with smaller bandwidths the receiver's impulse response becomes longer than the duration of a pixel, resulting in

(a) Original display image (Arial, 16 pixel/em, anti-aliased):

(b) AM demodulated analog video signal (VGA):

(c) AM demodulated 8 bit/pixel NRZ digital video signal:

(d) AM demodulated 10 bit/pixel TMDS video signal (DVI, HDMI):

(e) AM demodulated DisplayPort video signal:

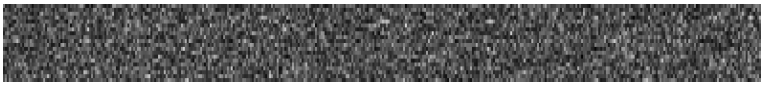


Figure 1.1: Different types of video signals as they appear to eavesdroppers after AM demodulation and rasterization. (The images above are the result of numeric simulations of line encodings and demodulation, and are therefore free of background noise.)

horizontal blurring of the reconstructed image.

While analog video signals may still be encountered today for backwards-compatibility reasons, flat-panel displays are generally designed to receive digitally-encoded pixels, and can similarly be vulnerable to video eavesdropping [3]. These synchronous serial interfaces have transfer rates of up to a few Gbit/s and typically three or four such serial interfaces (*lanes*) are used in parallel, often one for each primary colour. Early examples of such panel interfaces, e.g. *FPD-Link* [4] and *OpenLDI* [5], also known as *low-voltage differential signalling (LVDS)*, use a simple non-return-to-zero (NRZ) line encoding: the bits of the binary number that represents a pixel brightness appear directly, one after each other, as high or low voltages on the wire. Figure 1.1(c) shows what the same grayscale text image looks like if it was first NRZ encoded in a display device (least-significant bit first), and then AM demodulated by an eavesdropper. Different gray values result in different AM demodulation levels, due to the different amplitude spectra (Fourier transforms) of the respective bit patterns. While the resulting brightness mapping is not monotonic, it can preserve readable text.

NRZ-encoded LVDS video signals remain commonly used on flat cables *inside* devices, such as desktop monitors, laptops or TV-sets, to connect a display panel to a display controller on a separate printed circuit board. However, they are uncommon on external, user-accessible video-cable interfaces, where, for reasons of robustness, slightly more redundant and DC-free line encodings may be preferred.

Three standardised external video interfaces that use 8-to-10-bit line encodings have become widely used. The first two are the *Digital Video Interface (DVI)* [6] and the *High-Definition Multimedia Interface (HDMI)* [7]. Both use the same line encoding, known as *Transition Minimized Differential Signalling (TMDS)*, and can therefore be treated as the same signal type by eavesdroppers. Rather than transmitting the eight bits of pixel values directly, TMDS encodes them as one of several possible ten-bit symbols to ensure a DC-balanced output with a reduced number of bit transitions.

Figure 1.1(d) shows what the example image looks like to an eavesdropper with an AM demodulator if it was first TMDS encoded. Even though the encoding is more complex, and shows for example the DC-balancing mechanism cycling through different states in the uniform background, the text remains readable.

DisplayPort (DP) [8] is the third (and latest) major external digital video interface. It was first standardised in 2006 and is now commonly used, especially for higher-resolution desktop computer displays. Several aspects of the DisplayPort design suggest that its encoding is a much harder target for electromagnetic eavesdroppers. Firstly, image data is *scrambled* before transmission, to remove image-dependent spectral characteristics in electromagnetic emissions. This is not only helpful for passing radio-interference tests, but should also frustrate eavesdroppers using AM or FM demodulators. As Figure 1.1(e) shows, AM demodulating the (simulated) voltage signal found on a DisplayPort lane merely results in what looks like a random noise image. QAM or FM demodulation attempts lead to similar results. Secondly, due to the scrambling process, DisplayPort signals do not repeat at the frame rate, meaning that eavesdroppers cannot easily track the frame frequency or perform periodic averaging at that rate to improve the signal-to-noise ratio. Finally, the high bitrate (at least 1.62 Gbit/s) makes it hard to sample the signal with enough radio frequency bandwidth to be able to distinguish individual bits.

And indeed, unlike for DVI or HDMI, there have been no published video eavesdropping demonstrations that successfully target DisplayPort links. A 2019 study by Kubiak and Przybysz [9] targeted a DisplayPort monitor. The authors concluded that the only recognisable image they got was actually not from the DisplayPort cable, but from an LVDS link inside the monitor, i.e. after the DisplayPort signal had been decoded by a display controller into the NRZ-encoded signal that the panel used. They concluded that DisplayPort is "the safest in terms of electromagnetic security". A 2016 TEMPEST guidance document for designers of electronic voting machines [10], produced for the parliament of the Netherlands, compared the emissions security of many different video interfaces, and concluded already that eavesdropping on DisplayPort is "far more difficult, probably even impractical". It recommended the use of display panels with *embedded DisplayPort (eDP)* interfaces, to directly connect the graphics controller with the display panel, and thus avoid emissions from NRZ encodings.

DisplayPort is, however, vulnerable to eavesdropping using more sophisticated template-

based algorithms, which I present in this thesis for the first time. Rather than directly demodulating the received signal, the eavesdropper must first extract scrambler timing information and from it create a local copy of the scrambler synchronised with the target. They can then synthesise template signals for different colours in the image and classify pixel colours based on correlation with these templates. In the demonstrations presented in this thesis, I show that the algorithms I developed can be used to eavesdrop on a DisplayPort monitor at 8 m distance in realistic conditions using commercially available software-defined radio receivers.

1.1 History of electromagnetic eavesdropping

1.1.1 TEMPEST: a signals intelligence problem

Electromagnetic leakage of confidential data was first noticed as a problem in the First World War. Military telephone systems used single-wire cables, with ground as the return path, to halve cable cost and weight. It was soon discovered that the return current can be picked up and amplified by an adversary, and that it was possible to eavesdrop on these telephone communications from 100 yards away [11].

During World War II, British, American, and German intelligence agencies independently discovered that electromechanical *one-time tape* cipher machines leak signals that compromise data secrecy. One-time pad encryption is (assuming correct key management) mathematically unbreakable. However, plaintext could be recovered not from the ciphertext, but from radiated and conducted signals that were unintentionally produced by the machine. Plaintext processed by the "131-B2 mixer" used in American cipher machines could be reconstructed from radiated signals captured at 25 m distance [12]. The response to this finding was to require control of a 100-yard zone around the machine to ensure that the signal received by potential eavesdroppers is sufficiently attenuated. Similar problems were found in the British Rockex and German Siemens T43 machines [13].

Post-war, the problem was largely forgotten until the same issues were rediscovered in SIGTOT machines used to encrypt highly sensitive US communications in 1951. The same 131-B2 mixer was used in these machines, and the CIA discovered that it leaked plaintext via conducted emissions that could be used to read the message from "a quarter mile away" [12]. This discovery was a motivation for a research program on compromising emanations in the newly-formed NSA. The "General Studies on Radiation Suppression" program was established by 1954, with the cover name TEMPEST [14]. It took another two years until a "reasonably well protected" cipher machine, the KW-26, was successfully manufactured.

During the Cold War, compromising emanations were actively exploited by both sides of the conflict. The techniques were kept secret from other NATO members to ensure that

the American and British signals intelligence agencies had an advantage: in 1960, during negotiations on UK joining the European Economic Area, GCHQ tapped a teleprinter cable leaving the French embassy in London and recovered plaintext messages from conducted emissions [15].

Eavesdropping worries motivated governments to invest into shielded equipment and facilities and develop standards for such equipment. The first classified TEMPEST standard in the USA was NAG-1A in 1958, followed by a revised version FS-222 in the sixties. In 1970, *National Communications Security Emanations Memoranda* (NACSIM) were published, including test procedures and limits for electromagnetic emanations in NACSIM 5100. A long list of updates and other similar standards followed [16].

After the Cold War, the approach shifted away from relying on special shielded equipment due to decreased military budgets. A zone-based system where locations are classified based on the proximity to a potential eavesdropper (accounting for room shielding and building attenuation) was found to be more cost effective. The equipment certification requirements in higher-distance zones are less strict and can sometimes be passed by tested commercial off-the-shelf equipment, meaning that custom equipment is only needed in rare situations where confidential information must be processed in a location where the surroundings are not under control (e.g. a consulate) [17].

1.1.2 In academic literature

In the open literature, the possibility that unintended electromagnetic radiation might compromise security was first briefly mentioned in a RAND report on computer security and privacy in 1967 [18]. The first practical demonstration was in 1985, when Wim van Eck showed that an amplitude modulation receiver could be used to eavesdrop on an image displayed on a CRT monitor [2]. A number of papers also targeting CRTs followed [19–21], as well as a demonstration that RS-232 cables similarly leak data [22].

In the early 2000s, the first eavesdropping demonstrations on digitally encoded NRZ (e.g. LVDS) or TMDS (e.g. DVI or HDMI) signals appeared, using analog receivers [3, 23]. Later research on these targets mainly used digital *software-defined radio* receivers [24–31] (although some authors [23, 24] did not document the targeted type of video interface). One demonstration reports successful eavesdropping on a HDMI display from 80 m distance [28]. Other demodulation techniques successfully used include those for *quadrature amplitude modulation (QAM)* [26] and *frequency modulation* (FM) [29], as well as attempts to classify colours [26, 30].

Periodic averaging of the demodulator output at the frame rate is commonly used to improve the signal-to-noise ratio or to separate image signals from several sources. The frame rate is either adjusted based on image content [27, 31, 32], or automatically tracked with a narrow-band phase-locked loop following a harmonic of the pixel-clock frequency [26].

Recently, machine learning techniques have been applied as a post-processing tool to denoise the recovered image and improve text readability [33, 34]. Classifiers have also been used to show that an electromagnetic side channel leaks some data without directly reconstructing an image. Such demonstrations include classifying the contents of a LCD display (e.g. login screen, bank website, word processor, ...) from the reflection of a 24 GHz probing signal [35] and extracting smartphone PIN digits from signals generated by tapping the screen [36].

Targets other than computer displays are less common. Some demonstrations have used TV sets [37], tablets [38], and printers [39]. Electromagnetic radiation has also been used to profile program execution and detect deviations in behaviour [40], detect hidden cameras [41], and monitor keypresses on USB keyboards [42].

Proposed countermeasures include jamming [43,44], dithered patterns that are invisible to the user but cover the screen contents as seen by an amplitude demodulation receiver [45], and specially designed fonts. These *TEMPEST fonts* are created either by lowpass filtering glyphs of an existing font to eliminate high-frequency features that comprise most of the radiated energy [45, 46], or by making letter shapes similar so that they are difficult to distinguish by amplitude demodulation, which behaves as an edge detector for analog signals [47].

There is little published academic work targeting DisplayPort. The only attempt to reconstruct an image from radiated emissions used a standard TEMPEST receiver, and the authors conclude that the only usable signal was from LVDS circuitry inside the monitor [9]. Another demonstration successfully recovered image contents of several monitors, one of which used DisplayPort, from conducted power line emissions using an amplitude demodulation receiver [48]. They identify that the source of these conducted emissions is, however, likely monitor circuitry rather than the video interface. Research targeting exclusively DisplayPort has so far been limited to field strength measurements without any further discussion of image recovery [49, 50].

1.2 Motivation and scope

The work described in this thesis started as a shorter investigation of electromagnetic signals unintentionally emitted by DisplayPort interfaces. I had expected that the high signal bandwidth, compared to that of the software-defined radio receiver I used, and the use of scrambling and encoding would make image reconstruction impossible. I initially focused on understanding the DisplayPort specification and reverse-engineering implementation details to the point where I could create an image that is transmitted as a high-frequency square wave after scrambling and encoding. The resulting electromagnetic radiation would thus only consist of a small number of frequency components, and

measuring those would allow me to estimate the total emitted energy and whether DisplayPort emissions could be used as a covert channel. To my surprise, it turned out that I could not only recognise the signal emitted for specially designed images, but also that for any known byte sequence, leading to the initial image reconstruction algorithms I describe in Chapter 3 and motivating further research on the topic.

This thesis is the first in-depth treatment of unintentional DisplayPort emissions and signal processing algorithms that extract information from them. I describe multiple novel algorithms for timing extraction and image reconstruction, including the first successful electromagnetic eavesdropping demonstration targeting DisplayPort. While I provide some theoretical modelling and practical demonstrations, the focus is mainly on the design of these algorithms.

Since I did not have access to a shielded facility, I performed all experiments and demonstrations in a regular university building, without special measures to exclude outside radio sources. I therefore present their results as an example of what is practically possible and an approximation of the limits of described algorithms, rather than definitive measurements that can be used as a basis for equipment evaluation.

Overall, the main conclusion of the work presented here is that, contrary to existing opinion, scrambled signals such as DisplayPort are still vulnerable to electromagnetic eavesdropping. Despite the additional complexity, a periodic scrambling sequence followed by encoding is not enough to prevent image reconstruction, even if the receiver bandwidth is much lower than the interface bitrate.

1.3 Outline

The rest of this thesis is organised as follows:

- **Chapter 2** is a summary of relevant electromagnetic eavesdropping theory and current state of the art in the field. I start with a short introduction to electromagnetic compatibility and a model of the unintentional electromagnetic emissions of a differential signal such as DisplayPort, and then describe the hardware (antennas and radio receivers) used in my demonstrations. The chapter ends with an overview of the algorithms used for eavesdropping on older video interfaces (HDMI/DVI, FPD-Link, VGA) and a novel comparison of processing gains for different averaging methods.

- **Chapter 3** introduces the DisplayPort video interface and basic synchronisation and image reconstruction algorithms. I describe in detail parts of the DisplayPort standard that are relevant for the rest of this thesis, including reverse-engineered implementation details not specified by the standard. I then present two algorithms:

the first computes a reset model that approximately describes scrambler timing in the received signal, and the second then recovers the image, assuming that it consists of a known small set of colours.

- **Chapter 4** addresses the limitation of the second algorithm in Chapter 3 with an efficient colour enumeration algorithm. This consists of two main components as well: a PID-based scrambler tracking algorithm with improved scrambler timing error compared to the reset model, and an efficient template decomposition approach using properties of 8b/10b encoding which allows me to quickly compute a template for any RGB colour as a sum of seven (out of 61) sub-templates.

- **Chapter 5** applies these scrambler synchronisation and template generation techniques to image reconstruction. I extend the tracking algorithm with a phase correction step, which enables coherent averaging and thus a higher signal-to-noise ratio in the output image. Increased scrambler tracking accuracy also improves the horizontal resolution such that one pixel wide features are visible in the output.

- **Chapter 6** summarises the work described in the thesis, discusses possible countermeasures to electromagnetic eavesdropping on scrambled video interfaces, and finishes with suggested topics for further related research.

A repository containing a Julia implementation of the algorithms presented in the thesis, as well as IQ recordings of DisplayPort emissions used for evaluation and practical demonstrations, is available at `https://www.cl.cam.ac.uk/~de298/tempest-displayport/`.

Chapter 2

Background

2.1 Image format

Cathode-ray tube (CRT) monitors display images one pixel at a time, using an electron beam steered by a pair of magnetic deflection coils to scan over the display row by row. Since horizontal deflection cannot change instantly from the last pixel to the first one in the following row, video interfaces ensured that there is enough time to adjust the deflection by inserting a block of padding pixels at the end of each line (Figure 2.1). Several padding lines were similarly inserted at the end of each frame so that vertical deflection can reset to the first image line. These two padding regions are referred to as the horizontal and vertical *blanking interval* or *blanking region* in pre-DisplayPort video standards, and as *blanking periods* in DisplayPort.

Although modern displays do not use an electron beam, blanking regions, albeit shorter ones, are still used in newer video interfaces. They are separated from image data either by using a different encoding scheme, such as in DVI/HDMI, or by control symbols, such as in DisplayPort. Inside a blanking region, transmitted data is usually replaced

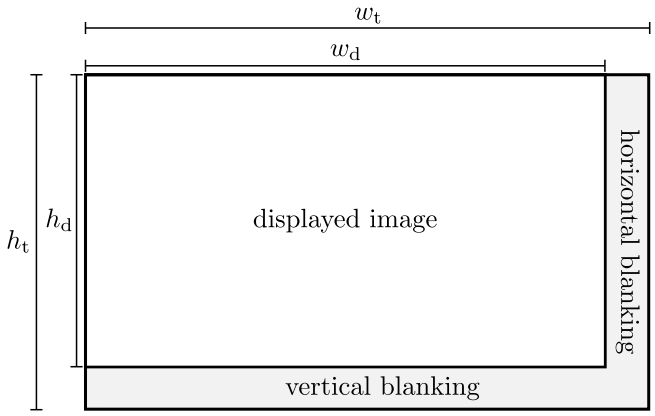


Figure 2.1: Diagram of a transferred image frame, showing padding bytes (gray) in blanking regions.

with zero-valued padding, or alternatively with additional low bitrate information such as audio samples.

Information about the image resolution, blanking regions and timing is collectively called the *video mode*, sometimes formatted in configuration files as a *modeline*. The video mode includes the following parameters:

- Displayed image width w_d and height h_d in pixels.
- Full image width w_t and height h_t in pixels, including blanking regions.
- Pixel clock rate f_p.
- The position, width, and polarity of the horizontal and vertical sync pulses (only relevant for analog interfaces).

Standard choices for these parameters and methods to compute them are defined in the VESA Coordinated Video Timings (CVT) Standard [51], or the older Display Monitor Timing (DMT) [52]. In this thesis, I will refer to video modes by their displayed dimensions $(w_\mathrm{d}, h_\mathrm{d})$ and frame rate f_v as $w_\mathrm{d} \times h_\mathrm{d}$ @ f_v without specifying the full dimensions, which will in all cases be those in the corresponding CVT mode.

The frame rate f_v is not defined directly, since it can be computed from the pixel rate and image dimensions as $f_\mathrm{v} = f_\mathrm{p}/(w_\mathrm{t} \cdot h_\mathrm{t})$. Although the frame rate is commonly shown rounded to zero or one decimal place, this is not an exact value. In CVT video modes, the pixel rate f_p is chosen instead to be a multiple of 0.25 MHz such that f_v is as close as possible to the desired frame rate. For example, the video mode 800×600 @ 60.3 fps with $w_\mathrm{t} = 1056$ and $h_\mathrm{t} = 628$ uses $f_\mathrm{p} = 40$ MHz, and so the exact frame rate is $f_\mathrm{v} = 40\ \mathrm{MHz}/(1056 \cdot 628) \approx 60.317$ Hz. Timing parameters for CVT video modes are chosen such that this calculated frame rate differs from the desired rate by at most 0.5%.

2.2 Electromagnetic compatibility

Changes in current flow in electronic circuits produce electromagnetic waves. This is usually not intended by the designer (except for transmitters), and can be undesirable if the emissions are strong enough to cause interference in nearby radio receivers or other devices. *Electromagnetic compatibility* (EMC) is an engineering discipline concerned with the design of systems that are compatible with their environment by ensuring that such unintended emissions are within acceptable levels. Although EMC compliance is a matter of limiting interference rather than preventing information leakage, a basic understanding of unintended electromagnetic emissions and design methods used to reduce them will be useful before we turn to DisplayPort eavesdropping. The following section is a high-level

introduction to the topic, including an overview of the relevant physical principles, EMC standards and design techniques used in computer monitors, and a mathematical model of emissions from a DisplayPort interface.

Apart from radiated emissions, the field of electromagnetic compatibility also includes radiated susceptibility, which is the design of circuits that can tolerate external interference, as well as conducted emissions and susceptibility, where interference signals are transmitted via cables (e.g. power line). This thesis focuses entirely on radiated emissions, as does most academic research on electromagnetic eavesdropping. Existing HDMI eavesdropping techniques have been shown to apply to conducted emissions as well, albeit with a different measurement setup [48].

2.2.1 Electromagnetic radiation

Electromagnetic phenomena are described by *Maxwell's equations*, which are a unified theoretical framework describing the electrical field $\boldsymbol{E}$ and the magnetic field $\boldsymbol{B}$. The study of electromagnetism is a large and complex topic, and here I will only provide a short introduction to the phenomenon of electromagnetic radiation. The interested reader is encouraged to consult one of many available books for an in-depth treatment [53, 54].

The magnetic field is produced by changes in the electric field and electric currents (Ampère-Maxwell law)

$$\nabla \times \boldsymbol{B} = \mu_0 \left(\boldsymbol{J} + \varepsilon_0 \frac{\partial E}{\partial t} \right) \tag{2.1}$$

where μ_0 and ε_0 are constants (magnetic permeability and electric permittivity of free space, respectively), and $\boldsymbol{J}$ is the density of current $\boldsymbol{I}$ through a point in space, i.e. an infinitesimally small perpendicular cross-section ∂S:

$$\boldsymbol{J} = \frac{\partial \boldsymbol{I}}{\partial S}. \tag{2.2}$$

Similarly, a change in the magnetic field produces an electric field (Faraday's law):

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t}. \tag{2.3}$$

The electric field flows from positive to negative charges, and so has non-zero divergence only in points where the charge density ρ is not zero (Gauss's law):

$$\nabla \cdot \boldsymbol{E} = \frac{\rho}{\varepsilon_0}. \tag{2.4}$$

The magnetic field forms closed loops and no magnetic charges exist (Gauss's law for magnetic fields):

$$\nabla \cdot \boldsymbol{B} = 0. \tag{2.5}$$

As a whole, these four laws are known as Maxwell's equations.

Intuitively, a changing magnetic field produced by a periodically varying current will induce a changing electric field, which in turn produces a magnetic field, and so on. Mathematically, we can derive the wave equation from Maxwell's equations by setting $\boldsymbol{J} = 0$ for space through which there is no current flow [53]:

$$\nabla^2 \boldsymbol{E} = \mu_0 \varepsilon_0 \frac{\partial^2 \boldsymbol{E}}{\partial t^2} \tag{2.6}$$

$$\nabla^2 \boldsymbol{B} = \mu_0 \varepsilon_0 \frac{\partial^2 \boldsymbol{B}}{\partial t^2}. \tag{2.7}$$

These equations describe the propagation of an *electromagnetic wave* through space, with speed $c = \sqrt{\mu_0 \varepsilon_0}$ equal to the speed of light.

The behaviour of the electric and magnetic fields in the *near field*, close to a radiating antenna, is complex. At sufficiently large distances, in the *far field*, the fields propagate as a spherical wave, with amplitude at distance d proportional to $1/d$. The boundary between the near and far field is often taken to be $\lambda/2\pi$, where λ is the wavelength of the emitted wave. This is the distance at which far-field behaviour becomes dominant, rather than a sharp cutoff, and some authors recommend a more conservative 3λ boundary [55] for a wire antenna, or $2D^2/\lambda$ for a surface antenna whose largest dimension is D.

2.2.2 Noise sources

The main sources of electromagnetic noise in a radio receiver can broadly be classified into three categories [58]:

- External natural sources, such as atmospheric gases, lightning discharges, and galactic noise from celestial bodies.
- External man-made sources, including electronic equipment, electrical machinery, power lines, and engine ignition.
- Thermal noise in the receiver due to the thermal energy of electrons in the receiving circuit.

Kuhn provides a summary of the expected noise levels in frequency regions of interest to an eavesdropper [56], based on survey data from the 2001 ITU-R Recommendation P.372 [57]. He concludes that in urban environments man-made noise is the most significant source at frequencies from 80 kHz upwards. Man-made noise dominates other external noise sources in all environments, with the exception of quiet rural sites, where it is exceeded by galactic noise above 4 MHz.

In urban environments, surveyed man-made noise power is greater than thermal noise. In rural environments, thermal noise exceeds external sources at high frequencies, starting

approximately at 150 MHz. In such environments, reducing thermal noise by cooling the receiving setup to very low temperatures can be a viable noise reduction approach.

It should be noted that the ITU man-made noise levels are based on data that is at least 30 years old; the most revision of the recommendation, from 2022, uses the same data [58]. Changes in technology, and in particular electronic devices using high clock rates, have changed the characteristics of man-made noise since then.

2.2.3 Standards and measurement

EMC regulations are the main motivation a device designer has for reducing electromagnetic emissions. These regulations impose a limit on the unintentionally emitted field strength, and a prospective design must be tested to show it meets the criteria before it can be marketed. Examples of such legislation are Directive 2014/30/EU (commonly known as the "EMC Directive") in the European Union [59], and the FCC Title 47 Part 15 regulations in the USA [60]. Most countries, with the USA as the main exception, base their EMC regulations for IT equipment on CISPR 22 [61], a standard by the International Electrotechnical Commission, or the newer CISPR 32 [62].

The regulations prescribe a standardised environment and procedure for measuring a device's emissions. The measurements are made either at an open-air site or in a semi-anechoic chamber, with the device 1 m above the floor ground plane and the receiver antenna at either 3 m or 10 m from the device (depending on the regulation and device type). Field strength measurements are made using a spectrum analyser.

A spectrum analyser is a receiver that measures the amplitude of the incoming signal's spectrum within a specified bandwidth and using a specified detector, over a wide range of frequencies. It contains an internal oscillator with tuneable frequency, which is swept over the measurement range and mixed with the received signal to select a part of the spectrum. The result is filtered to a fixed bandwidth, known as the resolution bandwidth. For EMC measurements, the resolution bandwidth must be at least 120 kHz. At frequencies above 1 GHz, the filtered signal's amplitude is measured directly; below 1 GHz, EMC standards specify a *quasi-peak detector*, a circuit designed to output a lower measurement for infrequently-occurring interference (a simplified example would be a capacitor charged by the input signal and slowly discharged through a resistor). Therefore, for example, the CISPR limit value of 70.8 µV/m in the 230–1000 MHz range means that the field strength measured using a quasi-peak detector in any 120 kHz band between 230 MHz and 1000 MHz must not exceed 70.8 µV/m.

Examples of FCC and CISPR limit curves are shown in Figure 2.2. These limits are chosen to constrain the noise emitted by the device below a level considered low enough to prevent interference with broadcast radio receivers in a 10 m radius, and thus considered compatible with the environment. EMC standards are not designed for confidentiality,

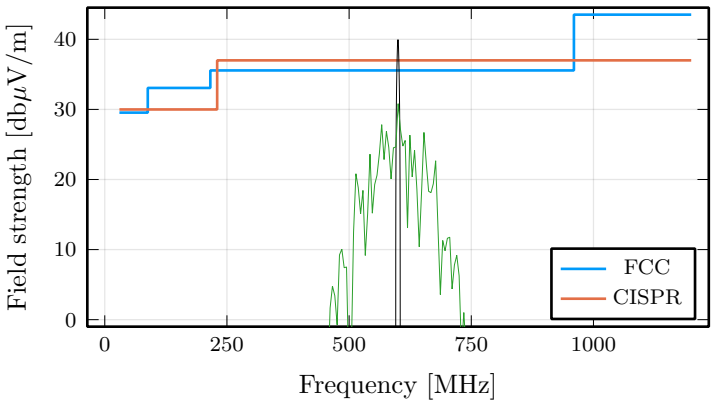


Figure 2.2: FCC and CISPR-32 radiated emissions limit curves for Class B (consumer) devices, at 10 m measurement distance. FCC limits were rescaled from the 3 m definition. Two example amplitude spectra, shown in green and black, have the same total power.

and signals below these limits may still contain information that can be extracted by an eavesdropper.

Emissions standards concerned with data confidentiality have long been the domain of militaries and intelligence agencies, and only high-level information is unclassified and publicly available. The relevant NATO standard is SDIP-27 (SECAN[1] Doctrine and Information Publication), which defines limit curves for three certification classes: A to C, from most to least strict. The accompanying SDIP-28 standard prescribes zoning procedures which classify rooms into zones based on attenuation, which can be provided by shielding or ensuring that adversaries cannot be close to the location. For example, only class A devices may be used in zone 0, where an adversary may have almost immediate access, while class B devices are allowed in zone 1, where adversaries can be kept at least 20 m away from the equipment (or the equivalent after accounting for building material attenuation) [64]. A third standard, SDIP-29, details installation requirements for such equipment for processing of classified information.

2.2.4 Design techniques

Many EMC design techniques exist, including signal design choices, shielding, circuit board design, filtering, and others, encompassing a range that is too broad to cover here. Below, I describe two that are used in DisplayPort and are relevant to the rest of this thesis: scrambling and differential transmission.

[1]Military Committee Communication and Information Systems Security and Evaluation Agency.

Scrambling

Emission limit curves define the maximum allowed field strength within any 120 kHz frequency band (spectrum analyser resolution bandwidth), measured at a specified distance from the device. This motivates the design of electronics where emissions are spread across the spectrum, since a wideband signal can have significantly higher total power than a sine wave or another narrowband signal while still staying below the limit (Figure 2.2).

Periodic signals are common in simple digital protocols, where short, repeating bit sequences can cause the unintended emissions to mainly consist of a small set of frequencies. As an extreme example, imagine a simple NRZ serial protocol such as LVDS, where the data bits are transmitted without any encoding at data rate f_b. When transmitting a stream of alternating zeroes and ones, the on-wire signal will be a square wave with frequency $f_\mathrm{b}/2$, and all emissions will be at odd harmonics $f_\mathrm{b} \cdot (2k+1)/2$ (for $k \in \mathbb{N}$) of f_b. More complex periodic data sequences, with a longer period T, similarly result only in emitted frequencies which are multiples of T^{-1}.

One common source of periodic data in computer display protocols are large image areas of same-colour pixels, such as the background colour in a displayed document. In a simple NRZ interface, where pixel encoding is stateless, the background has a one-pixel period. TMDS-encoded interfaces such as HDMI/DVI cycle through multiple encoded symbols for the same pixel value, and so the data period is a small multiple of the pixel period. Another source of repeating data are blanking periods, during which the interface transmits constant padding bytes if it is not sending other data (e.g. audio).

High-speed protocols such as Ethernet and DisplayPort *scramble* data before transmission to remove correlation between bits. The transmitter generates a pseudorandom scrambling sequence and combines it with the data using an invertible operation. For example, the DisplayPort scrambling sequence is generated by a linear-feedback shift register, and data bytes are replaced by their exclusive-or with the scrambler output. The receiver maintains a copy of the same scrambling sequence synchronised with the transmitter and undoes the scrambling operation before decoding. In DisplayPort, synchronisation is implemented using "scrambler reset" control symbols.

Although superficially similar to a stream cipher, scrambling is not encryption, and scrambled data can be decoded without knowing any secret information. The design of the pseudorandom generator and its parameters are publicly known and a part of the protocol specification.

Scrambling allows the interface designer to assume that transmitted data bits are uncorrelated. This is useful from an EMC standpoint because the emissions from transmitting a random bitstream will be spread over the entire spectrum, and short periods that would result in narrowband emissions are vanishingly unlikely. Sometimes, other device components can be simplified if the data can be assumed to be random, such as the clock

recovery circuit, for which randomness ensures frequent synchronisation opportunities on bit transitions.

Differential transmission

Differential transmission is a physical-layer design technique used to reduce electromagnetic interference emitted by high-speed interfaces such as HDMI and DisplayPort. The signal is transmitted over two separate paths such that the currents $I_1(t)$ and $I_2(t)$ on them are complementary: $I_1(t) = -I_2(t)$. The receiver recovers the signal as the difference $s(t) = I_1(t) - I_2(t)$. The two paths are usually a twisted pair of wires in a cable or parallel traces on a PCB, manufactured to minimise the distance d between them. In the limit where $d \to 0$, the EM fields generated by the two currents are equal in magnitude and have opposite signs. The total field is therefore zero, and the ideal differential pair does not radiate electromagnetic waves.

In practice, the distance d cannot be infinitely small, and the currents are not perfectly balanced. We can write the currents as a sum of a *common-mode current* $I_\mathrm{c}(t)$ and a *differential-mode current* $I_\mathrm{d}(t)$

$$I_1(t) = I_\mathrm{c}(t) + I_\mathrm{d}(t) \tag{2.8}$$

$$I_2(t) = I_\mathrm{c}(t) - I_\mathrm{d}(t) \tag{2.9}$$

where

$$I_\mathrm{c}(t) = \frac{I_1(t) + I_2(t)}{2} \tag{2.10}$$

$$I_\mathrm{d}(t) = \frac{I_1(t) - I_2(t)}{2}. \tag{2.11}$$

Approximate formulas for the magnitude of the radiated electric field, for a current with amplitude I and frequency f, can be derived by assuming that the wire length l is small compared to the wavelength and that the receiver is in the far field (i.e. that the wires can be treated as *Hertzian dipoles*) [55]. With these assumptions, the maximal radiated at distance L due to differential-mode current in two parallel wires is

$$|E_\mathrm{d,max}| = 1.316 \cdot 10^{-12} \frac{\mathrm{V}}{\mathrm{Hz}^2\ \mathrm{m}^2\ \mathrm{A}} \cdot \frac{I_\mathrm{d,max} f^2 l d}{L} \tag{2.12}$$

and that due to common-mode current is

$$|E_\mathrm{c,max}| = 1.257 \cdot 10^{-6} \frac{\mathrm{V}}{\mathrm{Hz\ m\ A}} \cdot \frac{I_\mathrm{c,max} f l}{L}. \tag{2.13}$$

Common-mode current generally contributes significantly more to the radiated field than a differential-mode current of the same magnitude. Common-mode radiation increases

linearly with f, while differential-mode radiation increases with f^2, and only exceeds common-mode radiation at the frequency f_e where

$$f_\mathrm{e} \cdot d \geq \frac{1.257 \cdot 10^{-6}\ \mathrm{Hz^{-1}\ m^{-1}\ A^{-1}}}{1.316 \cdot 10^{-12}\ \mathrm{V\ Hz^{-2}\ m^{-2}\ A^{-1}}} = 9.55 \cdot 10^5\ \mathrm{Hz\ m}. \tag{2.14}$$

For example, I measured the wire separation in a DisplayPort cable as slightly less than 1 mm. Rounding up to $d = 1$ mm, $|E_{\mathrm{d,max}}|$ exceeds $|E_{\mathrm{c,max}}|$ at $f_\mathrm{e} = 955$ MHz (at which point the Hertzian dipole approximation is no longer valid for typical cable lengths).

Additionally, $I_1(t)$ and $I_2(t)$ can differ due to a timing skew between the two wires, which causes signal edges to arrive at different times. In DisplayPort, the maximal inter-pair skew allowed by the standard is 50 ps (8% of the bit duration at 1.62 GHz). This causes a pulse, whose width is equal to the skew, in the common-mode current $I_\mathrm{c}(t)$. Different step responses of the line drivers due to transistor manufacturing tolerance can also cause a mismatch between the rising edge on one wire and the corresponding falling edge on the other. Both of these effects result in a larger $I_\mathrm{c}(t)$, and therefore a stronger emitted field, mainly at signal transitions.

In some serial interfaces, such as Ethernet and the MIL-STD-1553 avionic data bus, the transmitter is galvanically isolated from the cable using an isolation transformer. One transformer coil is driven by the transmitter, which induces a current in the other coil, connected across a differential pair. Such a circuit greatly reduces common-mode current, since the two wires in the pair no longer have separate line drivers. Common-mode current can be further reduced using a *common-mode choke*, a ferrite ring around which the two wires are wound in opposite directions. This has no effect on differential-mode current, but for common-mode currents it behaves as an inductor and therefore lowpass filter. These measures are, however, typically not used in video interfaces, and the DisplayPort standard only requires differential pairs to be AC-coupled.

We can model the radiated signal $E(t)$ as a sum of pulses with shapes $E_\downarrow(t)$ and $E_\uparrow(t)$ caused by the imbalance in $I_1(t)$ and $I_2(t)$ at falling and rising edges of $s(t)$, respectively. Write the wire voltage $s(t)$ as a purely digital signal corresponding to a discrete bit sequence $b[n] \in \{-1, 1\}$, sampled at rate f_b:

$$s(t) = b[\lfloor t \cdot f_\mathrm{b} \rfloor] \tag{2.15}$$

where any analog effects such as non-zero rise and fall time are absorbed into $E_\downarrow(t)$ and $E_\uparrow(t)$. Let $\mathbf{V}_\downarrow = \{n \in \mathbb{N} \mid b[n-1] > b[n]\}$ be bit indices of falling edges in $b[n]$, and $\mathbf{V}_\uparrow$ those of rising edges. The radiated signal is then

$$E(t) = \sum_{n \in \mathbf{V}_\downarrow} E_\downarrow(t - n f_\mathrm{s}^{-1}) + \sum_{n \in \mathbf{V}_\uparrow} E_\uparrow(t - n f_\mathrm{s}^{-1}) \tag{2.16}$$

or equivalently, in terms of a convolution with a Dirac pulse train,

$$E(t) = E_\downarrow(t) * \left(\sum_{n \in \mathbf{V}_\downarrow} \delta(t - n f_\mathrm{s}^{-1}) \right) + E_\uparrow(t) * \left(\sum_{n \in \mathbf{V}_\uparrow} \delta(t - n f_\mathrm{s}^{-1}) \right). \tag{2.17}$$

The two pulses $E_0(t)$ and $E_1(t)$ will have similar (but not necessarily identical) shape and opposite sign. The sign does not necessarily match the change in $s(t)$, and need not be consistent for separate differential pairs: the signal emitted by one differential pair might have opposite sign from that of a second one carrying the same data.

For the algorithms described in this thesis, it will be useful to approximate $E(t)$ as a signal containing all of $s(t)$, rather than just bit transitions. The pulse train in Equation 2.17 is the derivative of $s(t)$, and taking the derivative in the time domain is equivalent to multiplication by $2\pi \mathrm{j} f$ in the Fourier domain:

$$\mathcal{F}\left\{\frac{\mathrm{d}s(t)}{\mathrm{d}t}\right\}(f) = 2\pi \mathrm{j} f \cdot \mathcal{F}\left\{s(t)\right\}. \tag{2.18}$$

This transformation preserves the phase of $s(t)$ up to a constant $\pi/2$ rotation, and distorts the Fourier spectrum by a factor f. If we only receive frequencies $f_0 \leq f \leq f_1$, the distortion relative to the midpoint $(f_0 + f_1)/2$ is at most $(f_1 - f_0)/(f_1 + f_0)$, and we can consider $s(t)$ and $\frac{\mathrm{d}s(t)}{\mathrm{d}t}$ approximately equal in band-limited signals where $f_1 - f_0 \ll f_1 + f_0$, as is the case in this thesis.

2.3 Software-defined radio

In his early research on electromagnetic eavesdropping of CRT monitors, van Eck based the eavesdropping setup on a slightly modified broadcast TV receiver. Today, such experiments are normally done with less specialised equipment. Laboratory tests for equipment certification and research closely related to TEMPEST protection standards often use TEMPEST measurement systems [65], which are wideband AM/FM receivers designed to support frequency ranges and bandwidths needed for video eavesdropping. Most academic research, including this thesis, uses *software-defined radio (SDR)* technology instead, which allows for significantly easier experimentation.

A software-defined radio receiver is a general-purpose data-acquisition device for sampling a part of the radio spectrum and streaming the resulting time-series data to a computer, such that all further demodulation and decoding steps can be performed there in software. Such devices have become essential tools for emissions-security research, as they are not limited to particular modulation techniques. The main configuration parameters of an SDR are its tuning centre frequency f_c and its sampling rate f_s, which needs to be larger than the desired reception bandwidth B (e.g. $f_\mathrm{s} \geq 1.25 \cdot B$, Nyquist limit plus filter margin). The user chooses these parameters to acquire signals in the desired radio-frequency interval $[f_\mathrm{c} - B/2, f_\mathrm{c} + B/2]$.

The commonly used process for efficiently encoding a radio signal within such a frequency band in digital form is known as *IQ downconversion*, as it involves shifting those frequencies down by an offset $-f_\mathrm{c}$ to the *baseband* interval $[-B/2, B/2]$, where it can then be

Figure 2.3: USRP X300 software-defined radio receiver.

sampled efficiently with sampling rate $f_\mathrm{s} > B$. While the input antenna waveform $s_0(t)$ is real valued, the resulting time-domain samples will be complex numbers. This is because while the Fourier transform of a real-valued function is symmetric around 0 Hz (i.e. negative and positive frequencies carry the same information), this symmetry is broken by the $-f_\mathrm{c}$ downwards frequency shift, resulting in a complex-valued time-domain signal.

In mathematical terms, what an SDR receiver does to the input antenna waveform $s_0(t)$ is to downwards-shift its frequency spectrum by $-f_\mathrm{c}$ to obtain the baseband signal

$$s_\mathrm{d}(t) = \mathrm{e}^{-2\pi \mathrm{j} f_\mathrm{c} t} s_0(t) \tag{2.19}$$

by multiplying with a phasor rotating with frequency $-f_\mathrm{c}$ (where $\mathrm{j}^2 = -1$). This is then lowpass filtered to

$$s_\mathrm{f}(t) = \int s_\mathrm{d}(t-\tau) h(\tau) \mathrm{d}\tau \tag{2.20}$$

by convolving with the impulse response $h(t)$ of a lowpass filter that removes signals outside the frequency interval $[-B/2, B/2]$ (e.g. $h(t) = \mathrm{sinc}(tB)w(t)$, where $w(t)$ is some window function). The result is then sampled into a discrete-time sequence

$$s_\mathrm{r}[n] = s_\mathrm{f}(n/f_\mathrm{s}) \tag{2.21}$$

for sample indices $n \in \mathbb{Z}$. These recorded samples are typically represented as (real, imaginary) pairs of signed 16-bit integers or 32-bit floating-point numbers, and passed via USB, Ethernet or PCIe bus to the receiving computer. Such a data stream is also known as *IQ samples*, where *in-phase (I)* and *quadrature (Q)* are old-fashioned names for the real/cosine and imaginary/sine branches of a QAM signal path.

In all experiments described in this thesis, I used an Ettus USRP X300 software-defined radio receiver (Figure 2.3) [66]. The USRP is a platform that contains an analogue-to-digital converter (ADC) with 200 MHz maximum sampling rate, an FPGA, and multiple

(a) Yagi–Uda, Sinclair SY307-SF6SNM

(b) Log-periodic, Schwarzbeck VULSP 9111B

Figure 2.4: Antennas used in experiments and demonstrations described in this thesis.

high-speed interfaces. Analog signal processing (filtering and downconversion) is implemented as a replaceable daughterboard, which the user chooses based on frequency and bandwidth requirements. I used the UBX-160 daughterboard [67], which supports frequencies from 10 MHz to 6 GHz, with 160 MHz bandwidth. The daughterboard contains band selection filters, a local oscillator and mixer for downconversion, and low-noise amplifiers that can provide user-configurable gain between 0 dB and 31.5 dB.

At the sampling rate $f_s = 50$ MHz I used, the SDR outputs 50 million IQ samples per second, each represented as a pair of 32-bit floating-point numbers. The required data rate is therefore 50 MHz $\cdot 2 \cdot 4$ byte = 400 MB/s, or 3.2 Gbit/s. Gigabit Ethernet does not provide sufficient throughput, and so I used 10 Gigabit Ethernet over a fibre-optic cable (10GBASE-SR), either directly connected to a desktop computer's network card, or to a laptop via an Ethernet-to-Thunderbolt adapter.

2.4 Antennas

Since unintentionally radiated signals from video interfaces are weak and wideband, eavesdropping on them requires a broadband directional antenna. Directionality increases the received signal power relative to noise and interference from other sources, such as radio transmissions and other electronic devices. It is measured as *antenna gain*, usually expressed either in dBi relative to an isotropic antenna (an idealised model of a non-directive antenna), or in dBd relative to a half-wave dipole. Since the gain of a half-wave dipole is 2.15 dBi, a gain in dBi can be converted to dBd by subtracting 2.15 dB, and vice versa. Increasing the bandwidth allows the video eavesdropper to capture a larger fraction of the signal energy and improves time resolution.

The log-periodic antenna is a popular choice for EMC measurements that satisfies both properties. It is formed from several dipole elements, connected with alternating polarity. The element size decreases towards the tip of the antenna, forming a triangle where the

ratio of adjacent element lengths is constant. The bandwidth ranges approximately from the frequency at which the shortest element behaves like a half-wave dipole, i.e. where the wavelength is twice the length of the element, to that where the wavelength is twice the length of the longest antenna element. The directional gain of a log-periodic antenna is typically 6–11 dBi [68].

In my earlier experiments, I used a Schwarzbeck VULSP 9111B log-periodic antenna (Figure 2.4b). This is a calibrated antenna designed for reception in the 200–3000 MHz frequency range. In the 300–500 MHz range that I mainly used, it has 6–7 dBi directional gain. The large bandwidth makes the antenna suitable for initial exploration and characterising a potential target before a suitable eavesdropping frequency is known.

The Yagi–Uda antenna (often shortened to "Yagi") is also formed from parallel dipole elements, with three differences: only a single element is connected to the feed line, the elements are close in size, and they are spaced farther apart. It is designed for a narrow frequency range, where the wavelength is near one half of the element length. Unlike a log-periodic antenna, additional elements do not increase the bandwidth, but instead improve directional gain. Typical values range from 7 dBi to 20 dBi, depending on the number of elements.

In later experiments, I used a Sinclair SY307-SF6SNM antenna (Figure 2.4a). This is a Yagi–Uda antenna designed for 340–366 MHz, with $G_\mathrm{y} = 12.1$ dBi directional gain. Earlier measurements showed that there were no strong interfering signal sources in this range at my location, and that it was suitable for DisplayPort eavesdropping. Knowing this, a higher-gain antenna increased the range of my demonstrations.

Since the gains of the two antennas differ, experiments with the monitor at the same distance but with different antennas cannot be compared directly. For a 350 MHz signal, the log-periodic antenna has $G_\mathrm{l} = 6.4$ dBi gain. Replacing it with the Yagi would increase received signal power by $G_\mathrm{y} - G_\mathrm{l} = 5.7$ dB. As a crude approximation, let us assume free-space propagation where, at distance d from the source, power is proportional to $1/d^2$. Measurements made at distance d_l with the log-periodic antenna would, to an eavesdropper, look the same as ones made at distance d_y with the Yagi, where

$$\frac{d_\mathrm{y}^2}{d_\mathrm{l}^2} = 10^{5.7/10} = 3.72 = 1.92^2. \tag{2.22}$$

In other words, a successful eavesdropping demonstration using the log-periodic antenna suggests that the same would be possible at slightly less than twice the distance with the Yagi.

The directionality of the receiving setup can be further increased with a *phased array* of antennas, in which signals received from multiple antennas are added with phase offsets chosen to produce constructive interference in the desired direction. The signal-to-noise ratio can also be improved with more sophisticated signal conditioning, such as filters

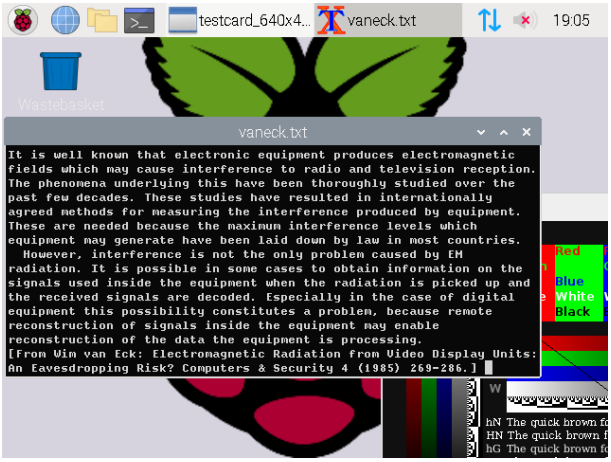


Figure 2.5: Image shown on the monitor used for demonstrations in this section.

that block known sources of interference (e.g. broadcast TV) and low-noise amplifiers. De Meulemeester et al. used such a setup, with a two-antenna phased array followed by two filtering and amplification stages, to successfully eavesdrop on an HDMI monitor from 80 m distance (compared to "around 10 metres" in previous research). While increasing signal-to-noise ratio in this way increases the viable attack range, it does not affect signal processing algorithms used after emissions have been captured by the SDR, and all work presented in this thesis would be equally applicable.

2.5 Introduction to video eavesdropping

Before discussing DisplayPort, it will be useful to start with a short introduction to signal processing algorithms used to eavesdrop on older video interfaces. I will use HDMI/DVI as an example since it is the most common TEMPEST research target, but similar techniques also apply to other non-scrambled interfaces such as VGA and FPD-Link.

Sample images in this section were reconstructed from a recording of the signals emitted by a Raspberry Pi model B+ showing the image in Figure 2.5, connected to a Dell 1704FPT monitor via a 1.8 m HDMI-to-DVI cable. This recording was made by Markus Kuhn using a Rohde & Schwarz FSV7 spectrum analyser and the VULSP 9111B log-periodic antenna, at approximately 1 m distance.

Let $s[n]$ be the received sampled IQ signal, centred at frequency f_c, with sampling rate f_s. If this is a recording of emissions from a simple NRZ interface, same-colour pixels in the displayed image will correspond to same signal content in $s[n]$. HDMI uses a TMDS encoder with multiple possible encodings for a byte, but there is only a small number of such encodings for a colour. Additionally, repetitions of the same value cycle through encodings in a fixed pattern.

At a high level, image reconstruction requires the eavesdropper to first extract timing

information (pixel, line, and frame rate) from $s[n]$, then demodulate it, optionally improve image quality by periodically averaging multiple frames, and rasterize the output to produce an image. Colours in the resulting image are not the same as the displayed image, and are instead a visualisation of a demodulation method, usually amplitude demodulation, that ideally provides good contrast.

2.5.1 Timing parameters

An HDMI interface transmits the entire image every frame, over three differential pairs for the red, green, and blue channel. Byte values are represented as 10-bit TMDS-encoded symbols. There is no padding other than blanking regions, and so the pixel rate f_p is the lowest rate for which the image can be transferred in a single frame period, with corresponding bitrate $f_\mathrm{b} = 10 f_\mathrm{p}$.

If the eavesdropper knows which video mode the target monitor uses, they can approximately compute the pixel rate from the frame rate f_v and total image dimensions $w_\mathrm{t} \times h_\mathrm{t}$ as $f_\mathrm{p} = f_\mathrm{v} h_\mathrm{t} w_\mathrm{t}$. This nominal rate will, however, not be accurate enough for image reconstruction. The frequency of the oscillator which provides clock timing will be slightly different from the desired value due to manufacturing errors and thermal effects, and the same relative error will be reflected in the pixel rate. An error in the pixel rate assumed by the eavesdropper will cause the reconstructed image to drift horizontally over time: if the assumed rate is f_p, and its true value is $\bar{f}_\mathrm{p}$, over a single frame the image will drift by d pixels, where

$$d = \frac{\bar{f}_\mathrm{p} - f_\mathrm{p}}{\bar{f}_\mathrm{p}} w_\mathrm{t} h_\mathrm{t}. \tag{2.23}$$

For example, for an 800×600 image where $(w_\mathrm{t}, h_\mathrm{t}) = (1056, 628)$, a $d < 1$ drift would require the error to be below 2 ppm (parts per million), an order of magnitude more accurate than the 30–50 ppm tolerance of typical oscillators. Periodic averaging requires even more accurate estimates: $d < 1/10 f_\mathrm{v}$, i.e. better than 0.01 ppm, is required to align frames for averaging over one second with at most 1/10 pixel misalignment.

Assuming that the displayed image is mostly static, the emitted signal will be periodic at the frame rate, with period f_v^{-1}. This period can be estimated from autocorrelation $R_{ss}[d]$ of $s[n]$:

$$R_{ss}[d] = \sum_n s[n] s[n+d]^*. \tag{2.24}$$

The autocorrelation magnitude will show large peaks at multiples of the frame period, as well as smaller ones at multiples of the line period (Figure 2.6). The frame rate can then be estimated from the position of the largest peak in R_{ss}, possibly restricted to a range of possible values $[d_0, d_1]$ for an assumed video mode:

$$f_\mathrm{v} \approx f_\mathrm{s} \cdot \left(\operatorname*{argmax}_{d_0 \le d \le d_1} |R_{ss}[d]| \right)^{-1}. \tag{2.25}$$

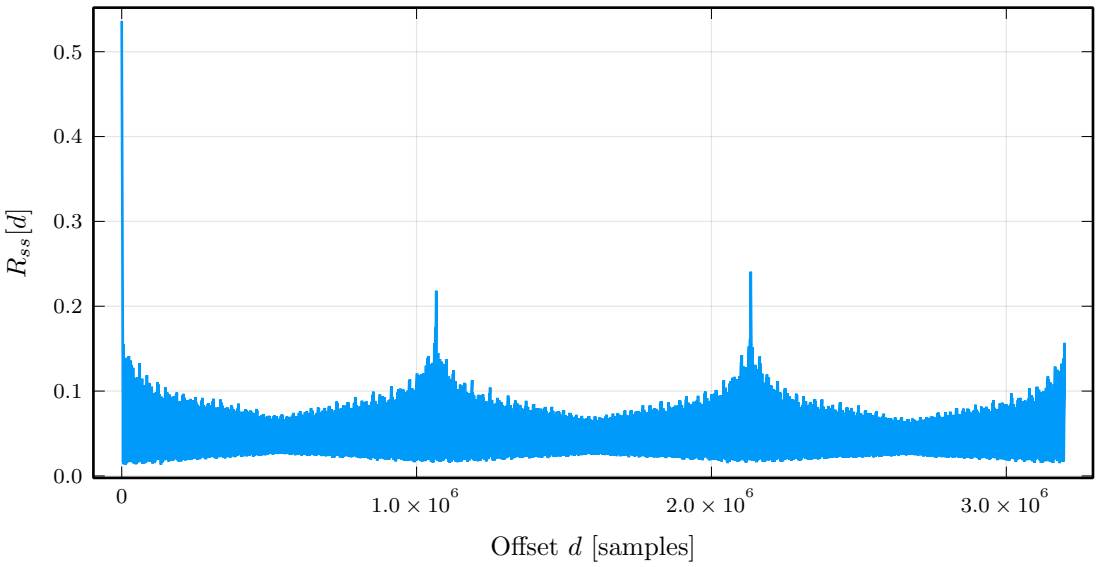


Figure 2.6: Plot of the autocorrelation $R_{ss}[d]$ of the received signal $s[n]$ containing emissions from an HDMI interface, showing peaks at multiples of the frame period.

The estimate can be further improved by searching for the k-th peak, located at $k \cdot f_{\mathrm{v}}^{-1}$, using this initial estimate for the frame period f_{v}^{-1}.

2.5.2 Rasterization

Next, $s[n]$ is resampled from f_{s} to f'_{s} to produce $s'[n]$, with the new sampling rate f'_{s} chosen so that one pixel period corresponds to an integer number of samples. The lowest rate that satisfies this property and does not discard data by downsampling is

$$f'_{\mathrm{s}} = \lceil f_{\mathrm{s}}/f_{\mathrm{p}} \rceil \cdot f_{\mathrm{p}} \tag{2.26}$$

where the length of each pixel in samples is

$$l = \lceil f_{\mathrm{s}}/f_{\mathrm{p}} \rceil \,. \tag{2.27}$$

A single image frame can then be reconstructed by taking $l \cdot w_{\mathrm{t}} \cdot h_{\mathrm{t}}$ consecutive samples in $s'[n]$, demodulating them, mapping the result to colours and displaying it as a $l \cdot w_{\mathrm{t}} \cdot h_{\mathrm{t}}$ rectangle.

The most common approach is amplitude demodulation, with grayscale output pixels whose brightness is proportional to $|s'[n]|$ or $|s'[n]|^2$. The simplest implementation of the entire sample-to-colour mapping would be

$$s'[n] \to \mathrm{Gray}\left(\frac{|s'[n]| - s_{\min}}{s_{\max} - s_{\min}}\right) \tag{2.28}$$

where $\mathrm{Gray}(b)$ is a grayscale pixel with brightness $b \in [0, 1]$, and $s_{\min}$ and $s_{\max}$ the minimum and maximum values of $|s'[n]|$ in the frame.

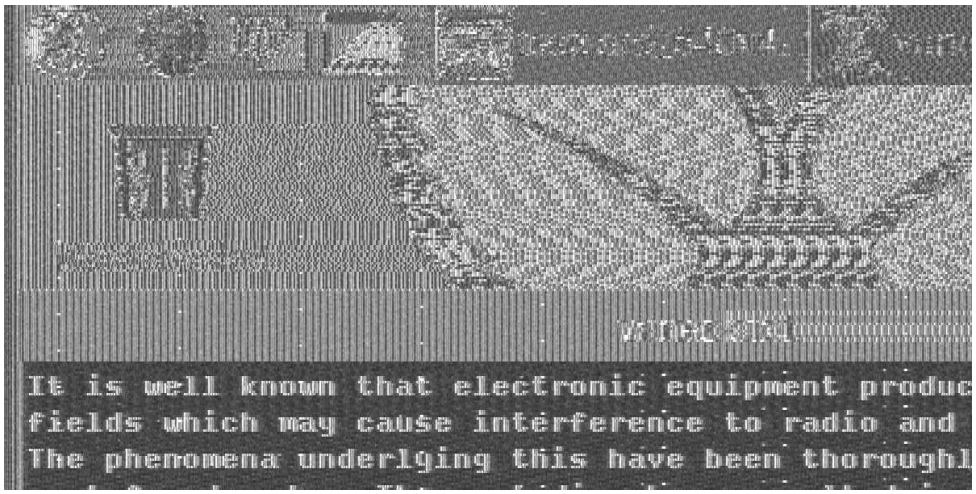


Figure 2.7: Sample from average of 60 amplitude-demodulated frames.

This mapping can unnecessarily discard a large part of the output dynamic range due to outliers in $|s'[n]|$, and produce images that are almost entirely comprised of dark shades of gray. For example, a short but strong signal burst unrelated to the eavesdropped image (e.g. a radio transmission or interference from another device) will increase $s'[n]$ in a small number of samples, and consequently increase $s_{\max}$. These samples will be displayed as a white dot or horizontal line, and the rest of the image will use a reduced level range.

The dynamic range can be increased by reducing the range $[s_{\min}, s_{\max}]$ of amplitudes that are mapped to possible pixel values in $[0, 1]$, and clamping all values below and above it to black and white, respectively. In my experience, choosing the thresholds such that a small fraction of values $|s'[n]|$ fall outside the range, between 1% and 5% on both sides, produced visually acceptable results (Figure 2.7). O'Connell recommends [26] choosing the thresholds based the sample average μ and standard deviation σ of $|s'[n]|$ as $s_{\min} = \mu - 3\sigma$ and $s_{\max} = \mu + 3\sigma$.

De Meulemeester et al˙ [29] showed that frequency demodulation of $s'[n]$ can also be used to eavesdrop on HDMI display units. They found that certain colour pairs that are difficult to distinguish using amplitude demodulation are more easily visible in FM-demodulated images, and suggested that using both methods could allow the eavesdropper to obtain more information and defeat some countermeasures.

O'Connell [26] demonstrated that both amplitude and phase information can be included in the same image using the hue-saturation-value (HSV) colour model. He represents amplitude as value (brightness), and phase as hue:

$$s'[n] \to \mathrm{HSV}\left(\angle s'[n], U, \frac{|s'[n]| - 3\sigma}{6\sigma}\right) \tag{2.29}$$

where $\mathrm{HSV}(h, s, v)$ is a pixel with hue $h \in [-\pi, \pi]$, saturation $s \in [0, 1]$ and value $v \in [0, 1]$ (implicitly clamping values outside this range), μ and σ are the sample mean and standard deviation of $|s'[n]|$, U is a user-defined constant, and $\angle s'[n]$ is the phase of $s'[n]$.

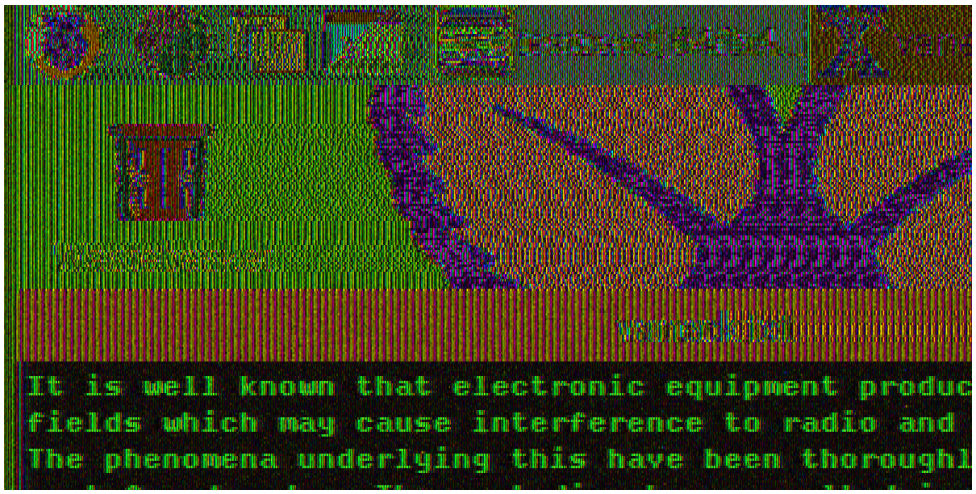


Figure 2.8: Sample from average of 60 phase-stabilised frames showing both amplitude as pixel value and phase as hue.

If such phase demodulation is directly applied to $s'[n]$, the output image shows a "rainbow-banding" effect across single-colour areas, making text difficult to read and unsuitable for coherent averaging. This happens because the phase of $s'[n]$ rotates due to the difference between HDMI clock rate f_H and SDR frequency f_c, with angular velocity $2\pi(f_\mathrm{H} - f_\mathrm{c})$. The rotation can be eliminated by downconverting the signal by $f_\mathrm{H} - f_\mathrm{c}$, i.e. multiplying each sample by a complex phasor

$$s'[n] \leftarrow s'[n]\mathrm{e}^{2\pi\mathrm{j}(f_\mathrm{H}-f_\mathrm{c})n/f'_\mathrm{s}}. \tag{2.30}$$

For shorter observation times, f_H can be estimated as a peak in the power spectral density of $s[n]$, and an example resulting image is shown in Figure 2.8. More accurate phase stabilisation, as needed for long-term coherent averaging, can be achieved using a tracking loop [26].

2.5.3 Periodic averaging

The noise level in the rasterized output can be reduced by *periodic averaging* of the signal for several image frames, assuming that the displayed image does not change during that period. One can average either the amplitude-demodulated signal (*noncoherent averaging*), or complex-valued samples before demodulation (*coherent averaging*). As I will show in this section, coherent averaging results in lower noise if the signal-to-noise ratio of the input is not very high and efficiently rejects interference from other signal sources.

I will first analyse an AWGN model of the received signal, and leave a discussion of non-Gaussian noise for later. Let $y_k \in \mathbb{C}$ be random variables describing IQ values that we wish to demodulate, e.g. signal samples for a particular image position after preprocessing

(resampling and phase alignment). We model y_k as independent, identically distributed random variables with mean $\mu \in \mathbb{R}^+$ and Gaussian noise with total variance σ^2, or $\sigma^2/2$ along the real and imaginary axis:

$$y_k \sim \left(\mu + \mathcal{N}\left(0, \sigma^2/2\right)\right) + \mathrm{j} \cdot \mathcal{N}\left(0, \sigma^2/2\right). \tag{2.31}$$

I assume that the samples have already been phase-aligned, and therefore choose μ to be positive real-valued without loss of generality.

The coherently averaged estimate of μ from n random variables y_k is

$$\psi_\mathrm{c} = \Re\left\{\frac{1}{n}\sum_{k=1}^{n} y_k\right\} \tag{2.32}$$

where the imaginary part is discarded since it only contains noise. This is an unbiased estimator:

$$\mathbb{E}\left[\psi_\mathrm{c}\right] = \mathbb{E}\left[y_k\right] = \mu. \tag{2.33}$$

Since the y_k are independent, averaging n values will decrease the variance by a factor $1/n$ (i.e. the standard deviation by $1/\sqrt{n}$). We can rewrite the estimator as an average of $\Re\{y_k\}$

$$\psi_\mathrm{c} = \frac{1}{n}\sum_{k=1}^{n} \Re\{y_k\} \tag{2.34}$$

each of which is distributed as $\mathcal{N}(\mu, \sigma^2/2)$, and so the variance of ψ_c is

$$\mathbb{V}\left[\psi_\mathrm{c}\right] = \frac{\sigma^2}{2n}. \tag{2.35}$$

The signal-to-noise ratio of an individual random variable y_k before demodulation is $\mathbb{E}\left[y_k\right]^2 / \mathbb{V}\left[y_k\right] = \mu^2/\sigma^2$, and so the processing gain of n-sample coherent averaging relative to the input signal is

$$G_\mathrm{c} = \frac{\mathbb{E}\left[y_k\right]^2}{\mathbb{V}\left[y_k\right]} \Big/ \frac{\mathbb{E}\left[\psi_\mathrm{c}\right]^2}{\mathbb{V}\left[\psi_\mathrm{c}\right]} = 2n \tag{2.36}$$

or in decibels

$$G_\mathrm{cdB} = 3\ \mathrm{dB} + 10\ \mathrm{dB} \cdot \log_{10} n \tag{2.37}$$

where the doubling comes from coherent demodulation discarding one half of the variance by eliminating the imaginary component. The coherent estimator ψ_c is optimal: it is the minimum variance unbiased estimator for μ [26].

Coherent averaging requires phase alignment that is accurate enough to ensure that the phase of y_k is stable over the averaging period. If such alignment is not available, we are limited to noncoherent averaging, which discards phase information and operates on the

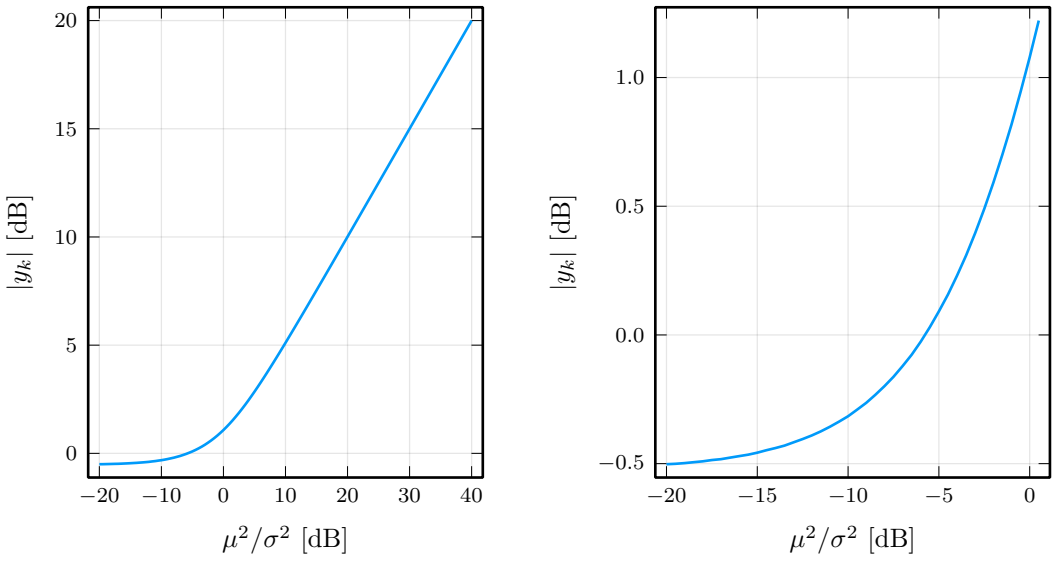


Figure 2.9: Plots of the expected value of amplitude-demodulated samples $\mathbb{E}[|y_k|]$, for varying input signal-to-noise ratio μ^2/σ^2.

magnitudes of y_k. The noncoherent (amplitude averaged) estimate of μ from n random variables y_k is

$$\psi_{\mathrm{n}} = \frac{1}{n}\left|\sum_{k=1}^{n} y_k\right|. \tag{2.38}$$

The amplitude-demodulated variables $|y_k|$ follow a Rice distribution with centre μ and scale σ:

$$|y_k| \sim \mathcal{R}(\mu, \sigma) \tag{2.39}$$

which is a generalisation of the Rayleigh distribution with a nonzero centre (for $\mu = 0$ the two are equivalent). Its mean and variance are

$$\mathbb{E}[|y_k|] = \sigma\sqrt{\frac{\pi}{2}} I_0\left(-\frac{\mu^2}{2\sigma^2}\right) \tag{2.40}$$

$$\mathbb{V}[|y_k|] = 2\sigma^2 + \mu^2 - \frac{\pi\sigma^2}{2} I_0^2\left(-\frac{\mu^2}{2\sigma^2}\right) \tag{2.41}$$

where I_0 is the order zero modified Bessel function of the first kind.

Since $\mathbb{E}[|y_k|] \neq \mu$, the noncoherent estimator is a biased estimator of μ. If the signal-to-noise ratio is very low ($\mu \ll \sigma$), the distribution tends towards a Rayleigh distribution with scale parameter σ, and $\mathbb{E}[\psi_{\mathrm{n}}] = \sigma\sqrt{\pi/2}$ only depends on the noise power. With increasing SNR, $\mathbb{E}[\psi_{\mathrm{n}}]$ becomes closer to the true mean μ, as shown in Figure 2.9.

For very high SNR ($\mu \gg \sigma$), the Rice distribution tends towards a Gaussian distribution with mean μ and variance $\sigma^2/2$. Similarly to coherent demodulation, amplitude demodulation of strong signals is not significantly affected by noise in the imaginary part of the

signal. If $y_k = (\mu + n_\mathrm{x}) + \mathrm{j} \cdot n_\mathrm{y}$, where $n_\mathrm{x}, n_\mathrm{y} \sim \mathcal{N}(0, \sigma^2/2)$ are the real and imaginary noise components, we can write the amplitude as

$$|y_k| = \sqrt{(\mu + n_\mathrm{x})^2 + n_\mathrm{y}^2} = \mu \sqrt{1 + \frac{2n_\mathrm{x}}{\mu} + \frac{n_\mathrm{x}^2 + n_\mathrm{y}^2}{\mu^2}} \tag{2.42}$$

and, approximating $\sqrt{1+a} \approx 1 + a/2$ for $|a| \ll 1$,

$$|y_k| \approx \mu \cdot \left(1 + \frac{n_\mathrm{x}}{\mu} + \frac{n_\mathrm{x}^2 + n_\mathrm{y}^2}{2\mu^2}\right) = \mu + n_\mathrm{x} + o(\mu^{-1}) \tag{2.43}$$

and so for large μ the noise term from n_y tends to zero.

From Figure 2.9, we can provide approximate thresholds for the two limiting cases:

- If the SNR of individual random variables y_k (i.e. samples) is above 25 dB, coherent and noncoherent averaging are approximately equivalent (less than 0.01 dB difference).
- If the SNR of individual y_k is below -15 dB, the result of noncoherent averaging is practically independent of the signal μ and therefore contains no information about the image.

SNRs that are of practical interest for electromagnetic eavesdropping are generally in the middle region between these limiting cases. We can compare the performance of the two estimators by evaluating their mean and variance directly (with a suitable approximation for the Bessel function), or via a numerical simulation. It is enough to consider the signal-to-noise ratio only for a single random variable y_k after demodulation; averaging n such values improves the SNR for both coherent and noncoherent demodulation by the same $10\ \mathrm{dB} \cdot \log_{10} n$.

The usual definition of the signal-to-noise ratio of a random variable x, $\mathbb{E}[x^2] / \mathbb{V}[x]$, is only applicable if x is zero-mean ($\mathbb{E}[x] = 0$), and cannot be used for amplitude-demodulated $|y_k|$. This is particularly obvious for the case $\mu = 0$, for which the expected value is

$$\mu_0 = \mathbb{E}\left[\left|\mathcal{N}\left(0, \sigma^2/2\right) + \mathrm{j} \cdot \mathcal{N}\left(0, \sigma^2/2\right)\right|\right] \neq 0 \tag{2.44}$$

and so the SNR would be non-zero even though no signal is available.

The *contrast-to-noise ratio* (CNR) is a more suitable measurement. This is a common image quality metric for instruments such as electron microscopes [69] and fMRI machines [70], which measures the contrast between two image features relative to noise. If A_0 and A_1 are average image levels for the two features, and σ^2 the noise variance, the CNR is commonly defined as

$$\frac{|A_1 - A_0|}{\sigma}. \tag{2.45}$$

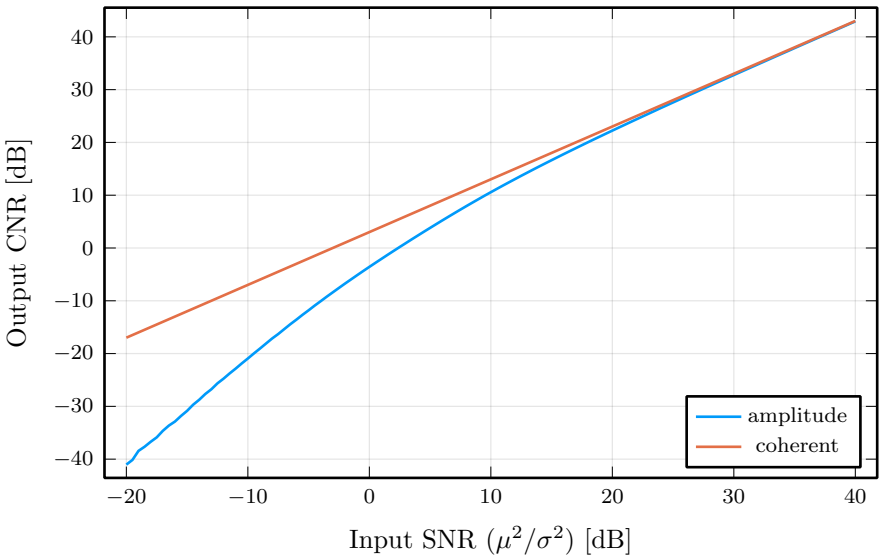


Figure 2.10: Plot of CNR of coherently and noncoherently (amplitude) demodulated random variables y_k, for varying input signal-to-noise ratio μ^2/σ^2.

Applying this approach, with the first feature $A_0 = \mu_0$ being only noise, and the second $A_1 = \mathbb{E}[|y_k|]$ the average signal amplitude, I define the CNR for an amplitude-demodulated image as

$$\mathrm{CNR} = \frac{(\mathbb{E}[|y_k|] - \mu_0)^2}{\mathbb{V}[|y_k|]} \tag{2.46}$$

where I have chosen to square the usual image processing definition to keep it analogous to a power-ratio SNR instead of an amplitude ratio. For a zero-mean signal where $\mu_0 = 0$, this definition of CNR is equivalent to SNR.

Scaling μ and σ by the same factor k scales both the numerator and denominator of Equation 2.46 by k^2. The amplitude demodulation CNR thus only depends on the received SNR μ^2/σ^2, and not on μ or σ individually. Figure 2.10 shows the numerically computed CNR for varying input SNR for both coherent and noncoherent demodulation. The difference between the two is the maximum processing gain from coherent averaging, relative to noncoherent averaging (Figure 2.11).

In a wideband radio signal, noise cannot be fully modelled as Gaussian, as it will contain radio transmissions and interference from electronic devices other than the eavesdropping target. Consider an extension to the AWGN signal model that includes an interference term with constant amplitude μ_i and varying phase $\varphi[k]$:

$$y_k \sim \left(\mu + \mu_\mathrm{i} \cos\varphi[k] + \mathcal{N}\left(0, \sigma^2/2\right)\right) + \mathrm{j} \cdot \left(\mu_\mathrm{i} \sin\varphi[k] + \mathcal{N}\left(0, \sigma^2/2\right)\right). \tag{2.47}$$

Assuming that the interference is not correlated with the signal we wish to extract μ, the phase will be uniformly distributed over the entire range $[-\pi, \pi]$:

$$\varphi[k] \sim \mathcal{U}(-\pi, \pi). \tag{2.48}$$

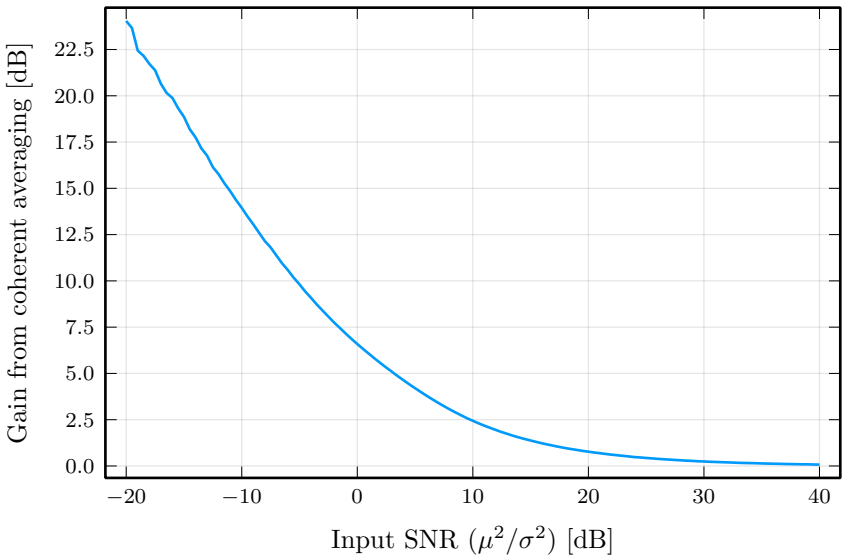


Figure 2.11: Plot of processing gain for coherent demodulation, compared to amplitude demodulation, as a ratio of CNR, for varying input signal-to-noise ratio μ^2/σ^2.

If these y_k are amplitude demodulated, the interference term biases the expected value similarly to Gaussian noise. If μ_i is large compared to μ, the noncoherent average will tend towards estimating the interference power rather than the desired signal μ, and amplitude averaging is again only suitable if the input SNR is sufficiently high.

The expected value of coherently averaged ψ_c is equal to the expected value of $\Re\{y_k\}$:

$$\mathbb{E}[\psi_\mathrm{c}] = \mathbb{E}[\Re\{y_k\}] = \mathbb{E}\left[\mu + \mu_\mathrm{i}\cos\varphi[k] + \mathcal{N}\left(0, \sigma^2/2\right)\right]. \tag{2.49}$$

By linearity of expectation:

$$\mathbb{E}[\psi_\mathrm{c}] = \mathbb{E}[\mu] + \mathbb{E}[\mu_\mathrm{i}\cos\varphi[k]] + \mathbb{E}\left[\mathcal{N}\left(0, \sigma^2/2\right)\right]. \tag{2.50}$$

By symmetry $\mathbb{E}[\mu_\mathrm{i}\cos\varphi[k]] = 0$, and so the expected value of the interference is zero, i.e. it will tend towards zero after coherent averaging, and ψ_c is still an unbiased estimator:

$$\mathbb{E}[\psi_\mathrm{c}] = \mu. \tag{2.51}$$

The main advantages of coherent demodulation and averaging are therefore its ability to reject interference other than Gaussian noise, and increased processing gain for low to moderate signal-to-noise ratios. While the processing gain of amplitude demodulation approaches that of coherent demodulation at very high SNRs, such strong signals are not encountered in practical electromagnetic eavesdropping attempts.

Chapter 3

Basic image reconstruction

Before discussing specific eavesdropping algorithms, it is necessary to understand how the target video interface represents and transfers image data. The first part of this chapter gives an overview of DisplayPort, focusing in particular on how images are converted into data packets and the on-wire representation of packet data. This information is mostly based on the VESA DisplayPort 1.2 standard [8]. The standard allows source designers some freedom to make implementation-specific choices for data packet size and padding details, and so this overview will also describe such implementation-specific choices, reverse-engineered from recordings of lane data.

The overview is followed by an eavesdropping attack that recovers an image transferred via DisplayPort from captured electromagnetic emissions. I first recover the timing parameters of the scrambling sequence and construct a copy of the scrambler synchronised with the target link. I then use this to build templates for a set of colours expected in the target image, and identify pixels matching these colours from the short-term cross-correlation between the recorded emissions and the templates. Such an attack would in particular be practical in applications where the eavesdropper has a-priori knowledge of the colours used in the user interface, as might be the case in specific applications, such as electronic voting machines.

These algorithms show that electromagnetic eavesdropping on DisplayPort is possible, and demonstrate a basic approach to two steps in such an attack: scrambler synchronisation and image reconstruction. Later chapters will cover improvements and more sophisticated algorithms which address limitations of this approach, by improving synchronisation accuracy, inferring likely colours without prior knowledge, decreasing noise and improving horizontal resolution in the recovered image.

This chapter begins with an overview of DisplayPort (Section 3.1). This is followed by a description of the attack, summarised in Section 3.2, which consists mainly of two algorithms: firstly I need to estimate some video timing parameters and synchronisation points (Section 3.3), which then enables me in a second step to classify the transmitted

pixel colours (Section 3.4) and recover substantial parts of the displayed image. Finally, I present practical demonstrations (Section 3.5).

3.1 DisplayPort

DisplayPort is a video interface designed for high-throughput unidirectional video data transfer. Its *main link* consists of four twisted wire pairs, known as *lanes*, using differential signalling.

Unlike DVI and HDMI, which base the clock rate on the video mode, a DisplayPort link has only a few possible bitrates: 1.62 Gbit/s, 2.7 Gbit/s, and in later versions of the standard also 5.4, 8.1, 10, 13.5, and 20 Gbit/s. The lowest bitrate that provides sufficient capacity for the current video mode is used, and the space between packets of image data is padded with *fill bytes*.

After inserting fill bytes and blanking periods, the data stream is scrambled by XORing it with the output of a 16-bit linear-feedback shift register and encoded using an 8b/10b encoder. The following subsections detail these steps, with Figure 3.1 providing an example.

3.1.1 Reverse engineering

The DisplayPort standard leaves details of the transfer unit padding algorithm to the implementer, and only specifies a high-level target (ratio of data bytes to transferred bytes). Since accurately modelling the on-wire data depends on this algorithm, I made several recordings of transmitted DisplayPort data for analysis and reverse-engineering. I examined two graphics controllers, both manufactured by Intel:

- Intel HD Graphics 520 (Gen 9, “Skylake”)
- Intel HD Graphics 4600 (Gen 7.5, “Haswell”)

I made the recordings using a Tektronix TDS7254B oscilloscope and P7330 differential probe, held to the connector pads on one of the ends of a DP cable (Figure 3.2a). Each recording contained the differential voltage of a single DisplayPort lane and covered around 750 image lines. A short sample of the recorded voltage can be seen in Figure 3.2b. The 2.5 GHz oscilloscope bandwidth is not high enough to estimate the rise time of the signal (and therefore the emitted bandwidth), but was sufficient to decode the data without error.

The padding details described in this thesis are therefore possibly specific to the Intel implementation. The eavesdropping algorithms in this chapter do not rely on any

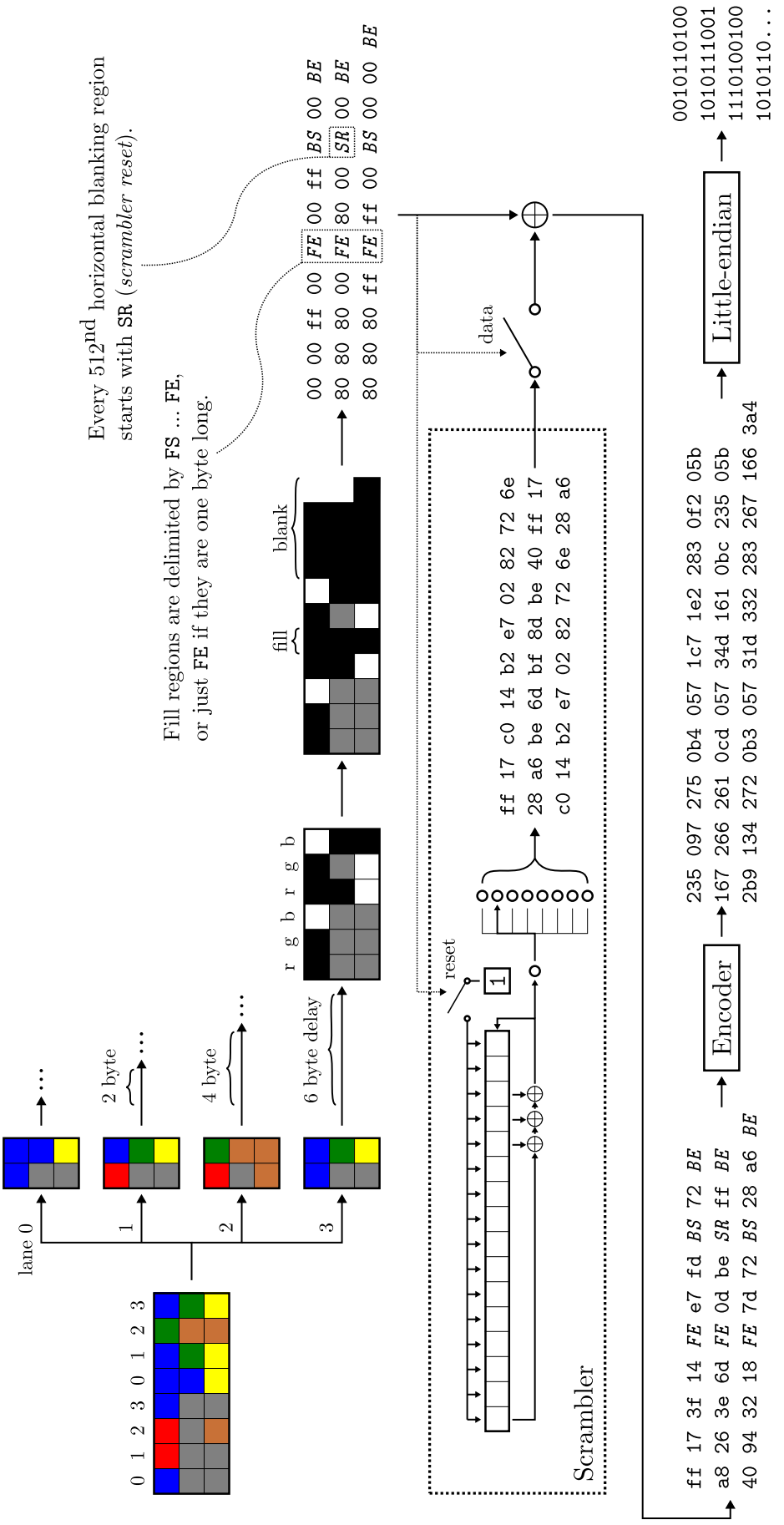
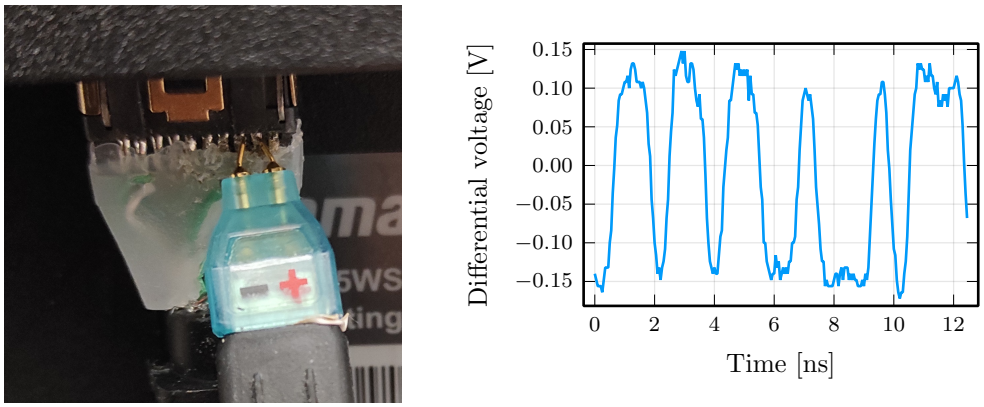


Figure 3.1: Overview of DisplayPort image framing and encoding, for a fictional video mode with eight pixels per line.



(a) **(b)**

Figure 3.2: (a) Differential oscilloscope probe held to the pads of a DisplayPort connector to measure differential lane voltage. (b) Single-shot recording of the differential voltage of a 1.62 Gbit/s DisplayPort lane, measured at the monitor end using an oscilloscope with 2.5 GHz bandwidth, sampled at 20 GHz, with a 0.3 pF active differential probe.

implementation-defined behaviour, and ones described in the following chapters could be trivially adapted to a different padding method.

In my experience, this is mostly a matter of testing small variations (e.g. rounding modes and precision) of a straightforward implementation to find one that matches observed data.

3.1.2 Data framing

The video mode used specifies the displayed image height h_d and width w_d, the vertical (frame) refresh rate f_v, and the total image dimensions h_t and w_t which include blanking periods. The latter are areas of zero-valued invisible pixels inserted at the end of each line and each frame to extend the $h_\mathrm{d} \times w_\mathrm{d}$ image to $h_\mathrm{t} \times w_\mathrm{t}$.

Since each pixel consists of three bytes (for the most common 24bpp RGB colour format), and each byte is encoded as ten bits, the minimal data rate required to transmit the resulting image is $30\ \mathrm{bits} \cdot h_\mathrm{t} \cdot w_\mathrm{t} \cdot f_\mathrm{v}$. The transmitter selects the lane bitrate f_b from the previously mentioned list of available values and the number of lanes $n_\mathrm{L} \in \{1, 2, 4\}$ to provide sufficient capacity, by ensuring that $30\ \mathrm{bits} \cdot h_\mathrm{t} \cdot w_\mathrm{t} \cdot f_\mathrm{v} \leq n_\mathrm{L} \cdot f_\mathrm{b}$, prioritising a lower bitrate f_b over a reduced lane count n_L.

If $n_\mathrm{L} > 1$, the image is split into n_L interleaved sub-images that will be transmitted on separate lanes. For example, for $n_\mathrm{L} = 4$, every fourth pixel from each line is assigned to the same lane, i.e. lane 0 contains pixels from columns 0, 4, 8, etc., while lane 1 contains columns 1, 5, 9, etc. Each pixel is then converted into bytes according to the colour format, so that all colour components of each pixel appear on the same lane.

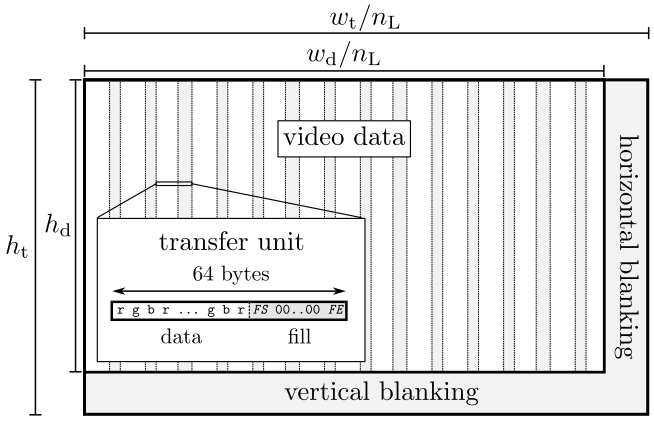


Figure 3.3: Diagram of data mapped to one lane for a DisplayPort video frame, including blanking periods and fill regions. Image data is shown in white, and zero-valued padding bytes are shown in gray. Image dimensions are given in pixels, before padding.

To meet the constant bitrate requirement, blocks (*fill regions*) of zero-valued fill bytes delimited by control characters are interleaved with the image data. Fill region placement does not respect pixel boundaries: fill bytes may be inserted in the middle of a pixel's three RGB bytes. A block of image data (except for the last block in each line) together with the following fill region is referred to as a *transfer unit (TU)*. All transfer units have the same implementation-defined length l_{TU}, which must be between 32 and 64 bytes. On the video interfaces I analysed, I only saw 64-byte transfer units.

The image blocks and fill regions start and end at the same byte offsets in all lines, relative to the first image byte in the line (see Figure 3.3). The average ratio of image bytes to transmitted bytes is

$$\rho = \frac{30 \text{ bits} \cdot h_t \cdot w_t \cdot f_v}{n_L \cdot f_b} \tag{3.1}$$

and the number of image bytes per transfer unit approximates $\rho \cdot l_{\mathrm{TU}}$. The last block of image data in each line is immediately followed by the (padded) horizontal blanking period, the length of which is chosen such that the total number of bytes sent per (padded) line is on average

$$n_p = \frac{f_b}{10 \frac{\text{bit}}{\text{byte}} \cdot f_v \cdot h_t}. \tag{3.2}$$

As n_p may in practice not be an integer, the length of the horizontal blanking period, and therefore the number of bytes transmitted for a line, can vary by one byte.

For example, for the 1920 × 1200 @ 59.95 fps video mode I used for demonstrations (Section 3.5), the total height including the vertical blanking period is $h_t = 1235$ lines, and the frame rate is $f_v = 59.95$ Hz. The lowest bitrate and lane count that provides the required bandwidth is $f_b = 1.62$ Gbit/s and $n_L = 4$. The average length of a padded line is therefore $n_p = 2188.06$ bytes. The fractional average length is met by choosing the

length of each padded line to be either 2188 or 2189 bytes, in a proportion that results in the correct long-term average.

The padded sub-image is read out line-by-line to produce the byte stream that will be transmitted over the corresponding lane. To provide resistance against bursty noise, a two-byte inter-lane skew is introduced: lanes 1, 2, and 3 are delayed by two, four, and six bytes, respectively, relative to lane 0. The resulting data is then scrambled, encoded, and transmitted to the receiver.

Implementation-specific details

Algorithm 1 DisplayPort line padding (assumed Intel implementation)

procedure PADLINE($d[], \rho, l_{\mathrm{TU}}$) ▷ Image data $d[]$, padding ratio ρ, TU size l_{TU}.
  $N \leftarrow 2^{23}$
  $M \leftarrow \lfloor N \cdot \rho \rfloor$
  $(n_{\mathrm{i}}, n_{\mathrm{f}}) = (0, 0)$
  for $b \in d$ **do**
    if $n_{\mathrm{i}} + n_{\mathrm{f}} + 1 \geq l_{\mathrm{TU}}$ **then**
      output $l_{\mathrm{TU}} - n_{\mathrm{i}}$ fill bytes
      $(n_{\mathrm{i}}, n_{\mathrm{f}}) \leftarrow (0, n_{\mathrm{f}} - (l_{\mathrm{TU}} - n_{\mathrm{i}}))$
    end if
    output image byte b
    $(n_{\mathrm{i}}, n_{\mathrm{f}}) \leftarrow (n_{\mathrm{i}} + 1, n_{\mathrm{f}} + N/M - 1)$
  end for
end procedure

The Intel video interfaces I analysed determine the number of fill bytes in transfer units using an approach similar to Bresenham's line-drawing algorithm [71] (Algorithm 1). The interface keeps track of the number of output bytes in the current image block n_{i} and the number of fill bytes that should be output n_{f}. The fill byte count is increased by $\rho^{-1} - 1$ after outputting an image byte. When $n_{\mathrm{i}} + n_{\mathrm{f}} + 1 \geq l_{\mathrm{TU}}$, the image block is terminated with $l_{\mathrm{TU}} - n_{\mathrm{i}}$ fill bytes, and the fill byte count is decreased by the same. In other words, fill region sizes are rounded up so that $n_{\mathrm{i}} + n_{\mathrm{f}}$ never exceeds l_{TU}.

Fractional byte counts are stored as rational numbers. The image-to-fill ratio ρ is rounded to M/N, where $N = 2^{23}$ and $M = \lfloor \rho \cdot N \rfloor$.

3.1.3 Scrambling

To flatten the frequency spectrum of electromagnetic interference generated, link data is *scrambled* using a maximum-length 16-bit *linear-feedback shift register (LFSR)* defined

	Bits	Encoded	
		RD = +	RD = −
00	000 00000	1101 000110	0010 111001
ff	111 11111	0111 001010	1000 110101
c0	110 00000	0110 000110	0110 111001
1a	000 11010	0010 011010	1101 011010
25	001 00101	1001 100101	1001 100101

Table 3.1: Examples of 8b/10b encoded values for positive and negative initial running disparity (RD). Binary values are shown here most significant bit first. Note that the encoding first considers the lower 5-bit block, and that the output is transmitted least significant bit first.

by the polynomial $x^{16} + x^5 + x^4 + x^3 + 1$. For each value (data byte or control symbol) transferred over the lane, the shift register is clocked eight times to produce a scrambler byte $\Xi[n]$. If the current value is a data byte $b[n]$, it is replaced by the exclusive-or $b[n] \oplus \Xi[n]$ of that byte and the scrambler byte $\Xi[n]$. Control symbols are not scrambled, but the shift register is still advanced. The same scrambling byte $\Xi[n]$ is applied to each lane, but delayed by 0, 2, 4, 6 bytes, respectively.

As it is defined by a degree-16 primitive polynomial, the scrambler has a period of $2^{16}-1 = 65535$ bits. Since $2^{16}-1$ is coprime with 8, the length of a byte, it takes eight shift-register periods for the bits to be placed at the same offset within a byte. Therefore, the resulting scrambler byte sequence has a period of $2^{16} - 1$ bytes.

Every 512 lines of image data, including the vertical blanking period, the scrambler is reset to the initial state `ffff` at the beginning of the horizontal blanking period. The receiver is notified of the reset by replacing a *blanking start (BS)* control character with *scrambler reset (SR)*. The 512-line period continues across frame boundaries and does not need to start on any particular line of a frame.

In the above 1920×1200 @ 59.95 fps example, the 512-line period is $512 \cdot n_\mathrm{p} \approx 1.12 \cdot 10^6$ bytes long, and therefore, between resets, the scrambling sequence repeats $512 \cdot n_\mathrm{p}/(2^{16} - 1) \approx 17.09$ times.

3.1.4 Encoding

Finally, the scrambled data is 8b/10b encoded, using the same encoder as in Gigabit Ethernet over fibre [72], and transmitted least significant bit first. The encoding ensures frequent transitions and also DC balance. This helps with clock recovery and allows links to be AC-coupled.

	Symbol	Encoded	
		RD = +	RD = −
SR	Scrambler Reset†	1101 000011	0010 111100
BS	Blanking Start†	1010 000011	0101 111100
BE	Blanking End	1110 100100	0001 011011
FS	Fill Start	1110 100001	0001 011110
FE	Fill End	1110 101000	0001 010111

Table 3.2: Control symbols used for transmitting video data, most significant bit first. (Symbols related to secondary data packets or content protection not shown.) †The SR and BS symbols may be replaced with four-symbol variants.

Each byte is encoded either as a DC-balanced 10-bit symbol containing five 0-bits and five 1-bits, or as an unbalanced symbol with four 0-bits and six 1-bits. In the latter case, the byte can alternatively be encoded as a secondary symbol with six 0-bits and four 1-bits. The choice is made depending on whether the output so far contains more 0- or 1-bits (negative or positive *running disparity*).

Let n_0 is the number of 0-bits output so far, and n_1 the number of 1-bits. The running disparity is then $RD = n_1 - n_0 - 1$, where -1 serves to break ties when the output so far is balanced. Initially, $RD = -1$. After encoding a byte with a balanced encoding, both n_0 and n_1 increase by 5, and the running disparity is unchanged. For bytes with unbalanced encodings, if $RD = -1$ the encoding with six 1-bits will be chosen, and so the new disparity will be $(n_1 + 6) - (n_2 + 4) - 1 = RD + 2 = 1$. Similarly, if $RD = 1$ the encoding with four 1-bits will be chosen, and the new disparity will be -1. Therefore, RD will always be ± 1 (usually written as $+$ or $-$), and the encoder only requires one bit of state.

Encoding a single byte whose encoding is balanced always leaves the running disparity unchanged, regardless of its value. Similarly, encoding a byte with unbalanced encodings always inverts the disparity. Therefore, changing the initial disparity will result in all unbalanced symbols being replaced by their respective alternate symbols.

To encode an 8-bit input, it is split into a 5-bit and a 3-bit block. These are encoded as a 6-bit and 4-bit output, respectively, using a pair of lookup tables. Each block can either have a single balanced encoding, with an equal number of zeroes and ones, or two unbalanced encodings that are complements of each other. Inverting the running disparity will therefore invert all bits in unbalanced output blocks, but leave balanced blocks unchanged (see Table 3.1). A complete table of encoded values is available in Appendix A.

Some 10-bit sequences that do not correspond to any 8-bit byte value represent *control symbols*, such as those inserted at the beginning and end of fill and blanking periods,

and the *scrambler reset* symbol. Control symbols used when transmitting video data are shown in Table 3.2.

3.1.5 An Intel-specific quirk

When examining oscilloscope recordings of DisplayPort lane data, I noticed that during a vertical blanking period, *blanking start* symbols are inserted six bytes earlier than when transmitting image data. If a *scrambler reset* replaces one of the misaligned symbols, the scrambling sequence used in the following 512 lines will be shifted by six bytes compared to behaviour if the bug was not present.

This contradicts the DisplayPort 1.2 specification, which states that they should be "inserted at the same symbol time during vertical blanking period as during vertical display."[1], and is likely a bug in the Intel implementation (possibly limited to certain generations). Mispositioned control symbols in the vertical blanking period do not change the image decoded by the monitor, since the entire blanking period is discarded before display.

Algorithms described in this chapter do not take this implementation-specific behaviour into account. I will return to the topic in Section 5.6.2, where I describe how these misaligned resets can be detected with accurate scrambler tracking and used to vertically align the reconstructed image.

3.2 Eavesdropping overview

Figure 3.4 provides an overview of the eavesdropping attack. It begins by recording the emissions of the target monitor using a software-defined radio receiver. Since practically available SDR sampling rates f_r (e.g., 50–200 MHz) are much lower than the DisplayPort link bitrate f_b, I can only capture a band-limited signal centred at a frequency of my choosing; experimentally, centre frequencies f_c around 400 MHz performed well at my location[2].

Before further processing, it was convenient to upsample the recording so that each sample corresponds to an integer number of transferred bits. In my case, this meant resampling to $f_\mathrm{s} = 1.62\ \mathrm{GHz}/32 = 50.625\ \mathrm{MHz}$.

I then process the recording to identify some time offsets $\mathbf{X}_\mathrm{o}$ at which the target lane scrambler is in its initial state (either because of a reset or a wraparound), by searching for predictable scrambled byte sequences. I fit a set of parameters describing the reset

[1] 2.2.1.1 Control Symbols for Framing: Default Framing Mode

[2] William Gates Building, 15 JJ Thomson Avenue, Cambridge CB3 0FD

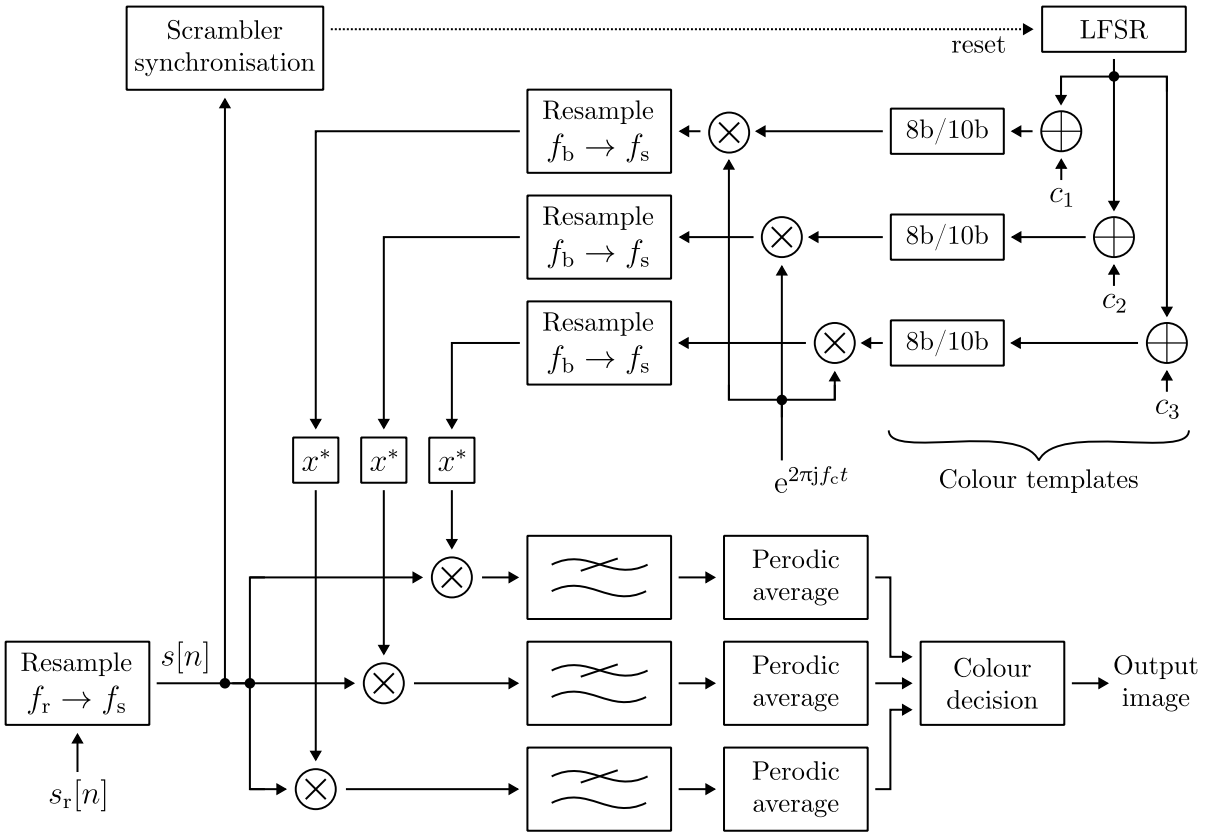


Figure 3.4: Overview of the image reconstruction attack that identifies three colours c_1, c_2, c_3 in the target image.

times to these offsets (*reset model*), allowing me to synthesise a scrambler synchronised with the recording.

I assume that the image mainly consists of a small set of colours $\mathbf{C} = \{c_1, c_2, \ldots, c_n\}$, i.e. this initial attack is more suited for text, line drawings or cartoon-like images, rather than continuous-tone photographs or video. For each of these colours, I synthesise a template signal based on the emulated scrambler. I then compute a short-term cross-correlation of each template and the recording, producing a correlation channel for each colour.

Each such channel is periodically averaged at the frame rate, to reduce noise, resulting in a single output frame. Such averaging can even be performed over multiple minutes, since the scrambler timing parameters can be estimated accurately enough to allow for precise long-term synchronisation.

Finally, all colour channels are combined into a single image. The image may need to be circularly shifted vertically to align it properly, and pixels recorded inside blanking and fill intervals can be removed before display.

3.3 Synchronisation

Let $\mathbf{X}$ be the set of time offsets at which the scrambler is in the starting state `ffff`. This will contain the start of every 512^{th} line, at which the scrambler resets, as well times between resets when the initial state repeats, due to the $2^{16} - 1$ byte period of the LFSR. With the simplifying assumption that all line lengths are equal, we can fully describe $\mathbf{X}$ by an initial offset x_0, a reset period R, and an LFSR period P:

$$\mathbf{X} = \left\{ x_0 + aR + bP \;\middle|\; a, b \in \mathbb{Z}, 0 \leq b < \frac{R}{P} \right\}. \tag{3.3}$$

The goal of the reset model is to reconstruct $\mathbf{X}$, so that I can align a synthesised signal template with the recording. I first extract from the $s[n]$ samples a set $\mathbf{X}_\text{o}$ of *observed* offsets at which the scrambler resets. Assuming that $\mathbf{X}_\text{o}$ is a subset of $\mathbf{X}$, up to a small error due to noise, I then find the best-fitting parameters $\boldsymbol{\theta} = (x_0, R, P)$.

3.3.1 Offset extraction

To identify time offsets at which the scrambler state is `ffff`, I rely on the presence of long blocks of consecutive zero bytes in the data prior to scrambling. Such blocks must occur regardless of the image, since the vertical blanking period is entirely filled with zeroes.

I first synthesise the expected signal $z_0[i]$ for N consecutive zero bytes, starting with a `ffff` scrambler state. The choice of N can be adapted to the expected length of the blanking period. Let the blanking period be N_b bytes long; if $N_\text{b} > 2^{16} - 1$, it will contain at least one position at which the scrambler is in the initial state, followed by at least $N = N_\text{b} - (2^{16} - 1)$ bytes. For the experiments described in this chapter, I optimise for the 1920×1200 @ 59.95 fps video mode and set $N = 11407$.

Let $\xi[i] \in \{-1, 1\}$ be the i^{th} bit of the 8b/10b encoding of the scrambler bytes following a reset (for $0 \leq i < 10N$), f_b the DisplayPort link bitrate, and f_c the centre frequency the receiver is tuned to. I first compute a template $\tilde{\xi}[i]$ sampled at f_b by downconverting $\xi[i]$, to match what the SDR does:

$$\tilde{\xi}[i] = \mathrm{e}^{-2\pi \mathrm{j} i \frac{f_\text{c}}{f_\text{b}}} \xi[i] \tag{3.4}$$

This is then downsampled to the receiver sampling rate f_s to obtain the template sequence $z_0[n]$ (for $0 \leq n < 10N f_\text{s}/f_\text{b}$).

Finally, I compute the cross-correlation $R_{z_0,s}[d]$ of z_0 and s as

$$R_{z_0,s}[d] = \sum_i z_0[i]^* s[i + d] \tag{3.5}$$

where * denotes the complex conjugate and the offset d ranges over the length of s.

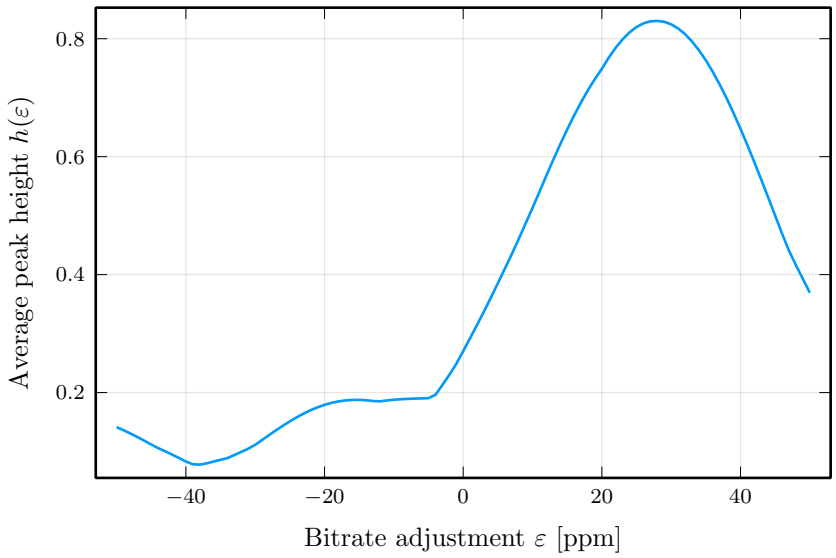


Figure 3.5: Plot of average peak height $h(\varepsilon)$ as a function of bitrate correction ϵ, for a recording made at 2 m antenna distance.

The offsets at which the scrambler state is `ffff` can be identified as peaks in $|R_{z_0,s}[d]|$. To eliminate false positives due to side peaks in the autocorrelation of z_0, I only consider local maxima that are the largest value within $\pm P/2$ samples, using an estimate of P computed from the link bitrate. I iterate through values of $R_{z_0,s}$ in descending order of magnitude, adding the offsets to the output set $\mathbf{X}_\mathrm{o}$ if they are at least $P/2$ samples away from all values currently in $\mathbf{X}_\mathrm{o}$. Since the scrambler overflow period is shorter than the length of the vertical blanking period, at least one identifiable overflow or reset occurs in each frame. I therefore terminate this procedure once $|\mathbf{X}_\mathrm{o}| \geq t_\mathrm{r}/f_\mathrm{v}$, where t_r is the duration of the recording $s[n]$ and f_v is the frame rate.

3.3.2 Bitrate adjustment

Small differences between the nominal bitrate f_b and the rate actually used by the interface, caused by oscillator manufacturing tolerance or thermal drift, accumulate over the length of $z_0[n]$, and the misalignment can significantly reduce the signal-to-noise ratio in the correlation $R_{z_0,s}[d]$. For example, if f_b differs from the nominal value by $\delta = 30$ ppm (a typical value I observed), the accumulated alignment error over N template bytes will be $10N \cdot \delta = 3.3$ bits.

The signal-to-noise ratio, and therefore also the distance at which synchronisation is successful, can be improved by first searching for f_b instead of using the nominal value. Let ε be the bitrate relative error, and define the average peak height $h(\varepsilon)$ for bitrate

$(1+\varepsilon)f_\mathrm{b}$ as

$$h(\varepsilon) = \sum_{k=0}^{\left\lfloor |s| \frac{f_\mathrm{v}}{f_\mathrm{s}} \right\rfloor} \max_{k\frac{f_\mathrm{s}}{f_\mathrm{v}} \le d < (k+1)\frac{f_\mathrm{s}}{f_\mathrm{v}}} |R_{z_0,s}((1+\varepsilon)f_\mathrm{b})[d]| \tag{3.6}$$

where $R_{z_0,s}(f'_\mathrm{b})[d]$ is template cross-correlation as in Equation 3.5, except with the template constructed using bitrate f'_b instead of f_b. Here I take only the highest peak in each frame period, since each frame will contain at least one in the vertical blanking period, but possibly no others at scrambler resets that happen inside the image region.

The average height will be maximal at ε for which $(1+\varepsilon)f_\mathrm{b}$ is the actual bitrate of the target interface (Figure 3.5). The updated bitrate used for offset extraction is then

$$(1 + \operatorname*{argmax}_{-\varepsilon_\mathrm{m} \le \varepsilon \le \varepsilon_\mathrm{m}} h(\varepsilon)) \cdot f_\mathrm{b} \tag{3.7}$$

where I set the range to $\varepsilon_\mathrm{m} = 50$ ppm and sample $h(\varepsilon)$ first with 10 ppm resolution, and then more finely with 0.5 ppm resolution near the candidate maximum.

3.3.3 Parameter fit

Informally, the best-fitting reset model is described by parameters $\boldsymbol{\theta} = (x_0, R, P)$ that produce a set of offsets $\mathbf{X}$ matching the observed $\mathbf{X}_\mathrm{o}$ as closely as possible. Let us define this formally as choosing these parameters, and coefficient mappings $a, b : \mathbf{X}_\mathrm{o} \to \mathbb{Z}$, as those that minimise the sum-of-squares error

$$\mathcal{E}(\mathbf{X}_\mathrm{o}|\boldsymbol{\theta}, a, b) = \sum_{x \in \mathbf{X}_\mathrm{o}} \big(x - (x_0 + a(x)R + b(x)P)\big)^2. \tag{3.8}$$

As in (3.3) above, $0 \le b(x) < \frac{R}{P}$ for all x.

I use an expectation-maximisation approach to find $\boldsymbol{\theta}, a, b$ that minimise $\mathcal{E}(\mathbf{X}_\mathrm{o}|\boldsymbol{\theta}, a, b)$ for a given $\mathbf{X}_\mathrm{o}$. The algorithm starts with an initial estimate for $\boldsymbol{\theta}$, and then alternates between updating the $a(x), b(x)$ and recomputing $\boldsymbol{\theta}$ estimates that minimise the error.

Initial values

If known, the target display's video mode can provide initial estimates for R and P. Let f_b be the lane bitrate used for the mode, f_v the frame rate, and h_t the total image height including the blanking period. The scrambler resets every 512 lines, so $R \approx \frac{512}{h_\mathrm{t} f_\mathrm{v}}$. The inner period P is independent of the line rate: $2^{16} - 1$ bytes, each encoded as 10 bits. Therefore, $P \approx \frac{10 \cdot (2^{16}-1)}{f_\mathrm{b}}$.

I do not attempt to estimate an initial x_0, and instead try multiple initial values for x_0, ranging from 0 to R in steps of $\frac{P}{2}$. For each of these I execute the parameter-fitting algorithm below, and keep the result that minimises $\mathcal{E}(\mathbf{X}_\mathrm{o}|\boldsymbol{\theta}, a, b)$.

Similarly, if the video mode is unknown, it can be identified by repeating the algorithm with initial R and P chosen for each likely video mode from a candidate list, and returning the one for which the algorithm below converges to the smallest error $\mathcal{E}(\mathbf{X}_\mathrm{o}|\boldsymbol{\theta}, a, b)$.

EM algorithm

Let $\boldsymbol{\theta}^{(n)}$ be the parameter values after n iterations, where $\boldsymbol{\theta}^{(0)}$ are the initial values described above. Similarly, let $a^{(n)}(x), b^{(n)}(x)$ be the best-fitting coefficients corresponding to $\boldsymbol{\theta}^{(n)}$.

I alternate between computing the coefficients

$$a^{(n)}, b^{(n)} = \operatorname*{argmin}_{a,b} \mathcal{E}(\mathbf{X}_\mathrm{o}|\boldsymbol{\theta}^{(n)}, a, b) \tag{3.9}$$

and the next estimate of parameters

$$\boldsymbol{\theta}^{(n+1)} = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{E}(\mathbf{X}_\mathrm{o}|\boldsymbol{\theta}, a^{(n)}, b^{(n)}) \tag{3.10}$$

until convergence.

I compute the solution to (3.9) by assigning the values separately for each $x \in \mathbf{X}_\mathrm{o}$:

$$a(x) \coloneqq \left\lfloor \frac{x - x_0}{R} \right\rfloor \tag{3.11}$$

$$b(x) \coloneqq \left\lfloor \frac{x - x_0 - a(x)R}{P} + 0.5 \right\rfloor \tag{3.12}$$

As a special case, if the offset is near a scrambler reset and so $b(x) = \left\lfloor \frac{R}{P} \right\rfloor$, incrementing $a(x)$ by 1 and setting $b(x) = 0$ can result in a lower error.

By taking partial derivatives with respect to x_0, R, P, we can find the solution to (3.10) by solving a system of linear equations:

$$\sum_{x \in \mathbf{X}_\mathrm{o}} \left(\begin{pmatrix} 1 \\ a(x) \\ b(x) \end{pmatrix} \begin{pmatrix} 1 \\ a(x) \\ b(x) \end{pmatrix}^\mathsf{T} \right) \begin{pmatrix} x_0 \\ R \\ P \end{pmatrix} = \sum_{x \in \mathbf{X}_\mathrm{o}} \begin{pmatrix} x \\ a(x) \\ b(x) \end{pmatrix} \tag{3.13}$$

3.4 Image reconstruction

In the next step, I assume that the image displayed on the screen mostly contains a small set of colours $\mathbf{C} = \{c_1, c_2, \ldots, c_n\}$, as is the case for text and line drawings (e.g. for demonstrations in Section 3.5, $n \leq 16$). For each colour c_k, I construct a pair of positive- and negative-disparity templates $z_k^+[i]$ and $z_k^-[i]$ containing the expected signal from a DisplayPort lane that only transmits repetitions of c_k. These are constructed in the same

way as the template used for synchronisation, by scrambling and 8b/10b encoding the data, then downconverting and resampling (see Section 3.3.1).

The difference between the two templates is the initial disparity of the 8b/10b encoder, which is set to positive when generating z_k^+ and negative for z_k^-. Since a change in disparity propagates through the entire encoded stream, this ensures that one of the two templates matches encoded image data, regardless of the running disparity after encoding preceding pixels.

Next, I iterate through the received signal samples $s[n]$, keeping track of the number i_s of samples since the last scrambler reset to align the templates with the signal. This value increases alongside n, and is reset to zero at predicted resets where $n \in \mathbf{X}$. For each colour k, I compute the cross-correlation (dot product) of a window of w samples in s and both templates, and compute a score $u_k[n]$ by taking the template with the higher correlation:

$$u_k[n] = \max_{z \in z_k^+, z_k^-} \left\{ \left| \sum_{l=0}^{w-1} s[n+l] z[i_\mathrm{s}+l]^* \right|^2 \right\} \tag{3.14}$$

The choice of window length w poses a tradeoff: longer windows reduce the noise level in the final image, but result in a horizontally blurred output in which narrow features cannot be distinguished. For the experiments presented in this chapter, I chose $w = 5$.

The score $u_k[n]$ for each colour is averaged over all frames, using a frame length (in samples) $(h_\mathrm{t} \cdot R)/512$ computed from the scrambler timing parameter R and the video mode's total height h_t.

If there are multiple colours of interest, I assign the highest-scoring colour $\mathrm{argmax}_k\, \bar{u}_k[i]$ to each pixel (*voting mode*). If the target image mostly consists of a single background and foreground colour, c_1 and c_2, for example if it mostly contains text, the scores can alternatively be mapped to grayscale values by taking (after periodic averaging) the difference $\bar{u}_1[i] - \bar{u}_2[i]$, and normalising the resulting values into pixel brightness values in $[0, 1]$ (*silhouette mode*).

At this stage, the image is still cyclically shifted by an unknown offset. I first shift it such that the sample at time x_0 maps to right after the last pixel in a line, since the scrambler is always reset at the beginning of a horizontal blanking period. This ensures horizontal image alignment. Vertical image alignment can be adjusted manually, or using an automated algorithm described in Section 5.6.2.

3.5 Experimental results

Next, I first show measurements of the success rate and accuracy of the synchronisation algorithm from Section 3.3. I then present practical demonstrations of the image

reconstruction algorithm using three displayed target images: one containing text, both with and without anti-aliasing, and with colours that are particularly difficult to distinguish for this technique (Section 3.5.3); a presentation slide that an eavesdropper might be interested in (Section 3.5.4); and a cartoon image with a larger number of colours (Section 3.5.5).

3.5.1 Setup

The eavesdropping target for all experiments described below was a laptop with Intel Skylake GT2 graphics controller, connected via a miniDP-to-DP adapter and 1.5 m DisplayPort cable to an iiyama ProLite XUB2495WSU LCD monitor. I also tested the same attacks targeting instead a desktop computer with Intel Xeon E3-1200 graphics controller, for which it worked without modifications. For the image-reconstruction demonstrations (Section 3.5.3 and later) the target used the highest-resolution video mode supported by the display: 1920×1200 @ 59.95 fps.

The emissions for image recovery demonstrations were recorded using the Schwarzbeck VULSP 9111B log-periodic antenna directly connected via a 4.6 m long RG-213/U coax cable to an Ettus USRP X300 software-defined radio receiver with Ettus UBX-160 daughterboard. The receiver was tuned to a centre frequency of $f_\mathrm{c} = 400$ MHz, with sampling rate $f_\mathrm{r} = 50$ MHz, streaming complex float32 values to a MacBook Pro via a 10GBASE-SR optical-fibre Ethernet-to-Thunderbolt adapter.

For each image-recovery demonstration shown, I made ten two-second recordings. I chose two seconds because, in my experience, that duration was sufficient to provide reliable scrambler synchronisation. Significantly shorter recordings resulted in insufficient data for accurate parameter estimation, whereas processing much longer recordings would have exceeded the RAM available in the computer I used for analysis. Scrambler synchronisation was accurate enough to align all resulting images across the ten recordings (based on recording timestamps) and produce a single averaged output. All recovered images shown are such averages, each computed from a total of 20 seconds of recorded emissions.

Recordings used for the image-recovery demonstrations were made in a meeting room of a university building, without any measures to reduce interference from other nearby devices. A photo of the setup (antenna and target system) can be seen in Figure 3.6.

It should be noted that in this demonstration, the cable was (not deliberately) oriented perpendicular to the antenna, and that for a straight cable this receiver position maximises directional gain. In an anechoic environment, this directionality would significantly impact the received signal power, which would be minimal if the cable is oriented in the same direction as the antenna. In my experience, this difference is less pronounced in indoor environments, where multipath propagation reduces the impact of directionality.

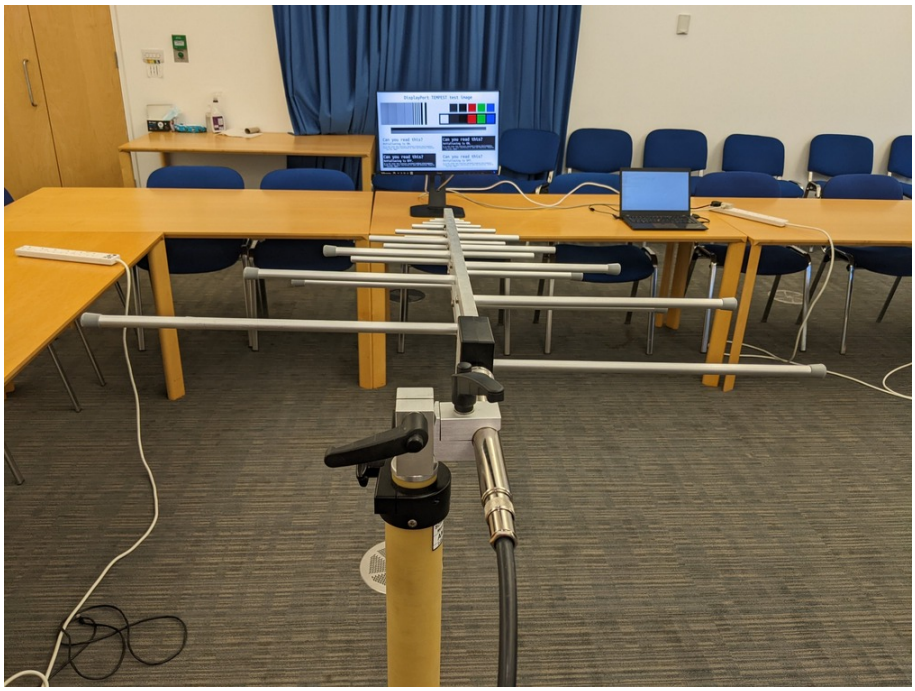


Figure 3.6: Measurement setup with 2.0 m distance between the tip of the eavesdropping antenna and the surface of the targeted display, connected via a 1.5 m DisplayPort cable (white) and short miniDP adapter cable (black) to a laptop.

Changing cable orientation would visibly affect the noise level in the reconstructed image, but not to the point where, for example, previously legible text would be rendered unreadable.

The evaluation of the synchronisation algorithm was carried out in a hallway in the same building, with the same receiver hardware except for the antenna, which was the Sinclair SY307-SF6SNM Yagi-Uda antenna, and the centre frequency $f_c = 350$ MHz.

3.5.2 Synchronisation

I evaluated the synchronisation accuracy of the algorithm from Section 3.3 for antenna distances between 2 and 16 metres, for three video modes (1920 × 1200 @ 59.95 fps, 1920 × 1080 @ 60 fps, and 800 × 600 @ 60.3 fps). For each distance and video mode, I made 50 two-second recordings, for each of which I estimated the scrambler timing parameters $\boldsymbol{\theta}$, the mean-square error $\mathcal{E}(\mathbf{X}_o|\boldsymbol{\theta}, a, b)$ after convergence of the EM algorithm, and the most likely video mode (i.e. the mode that provided the initial R and P values that led the EM algorithm to the lowest $\mathcal{E}(\mathbf{X}_o|\boldsymbol{\theta}, a, b)$) out of 32 candidate modes.

I considered synchronisation successful if $\sqrt{\mathcal{E}(\mathbf{X}_o|\boldsymbol{\theta}, a, b)} < 2$ samples, which is accurate enough to reconstruct an image. I chose this threshold after observing the distribution of errors: for almost all recordings, it was either under 2 or above 100 samples; there were none between 2 and 20.

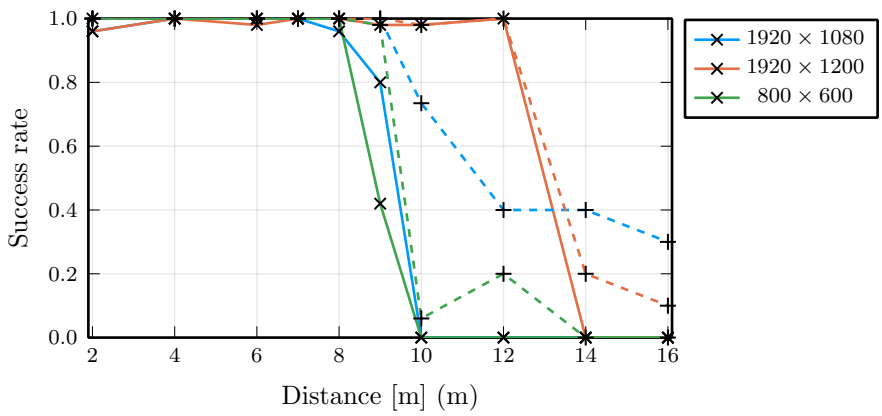


Figure 3.7: Fraction of recordings where synchronisation was successful (solid lines) and where the mode was correctly identified (dashed lines), depending on the antenna distance.

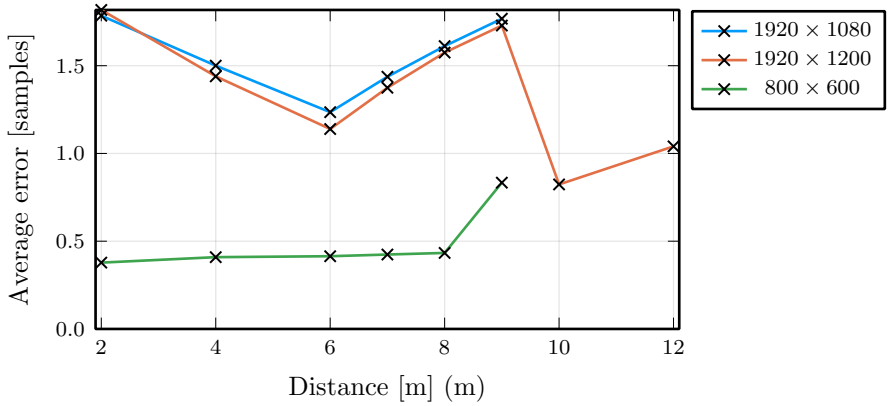


Figure 3.8: Root-mean-square error $\sqrt{\mathcal{E}(\mathbf{X}_\mathrm{o}|\boldsymbol{\theta}, a, b)}$ averaged over recordings where synchronisation was successful, depending on the antenna distance.

Figure 3.7 shows that up to eight metres away, the video mode was always correctly identified, and synchronisation was successful in over 90% of cases. With the target farther away, performance drops off quickly. This change happens at the distance where the signal-to-noise ratio is not high enough to reliably detect correlations between the signal and expected scrambler data, meaning that some entries in the obtained $\mathbf{X}_\mathrm{o}$ are not actual scrambler resets.

Figure 3.8 shows that averaged root-mean-square synchronisation error after successful synchronisation does not increase with distance significantly. This shows that the main limiting factor for synchronisation is the identification of offsets $\mathbf{X}_\mathrm{o}$ at which the scrambler resets. After that, added noise does not impact accuracy significantly.

Note that the best-case mean-square error depends on the target video mode. This is because the reset model is based on the simplifying assumption that the delay R between scrambler resets is constant. In practice, this assumption is violated in two ways. First,

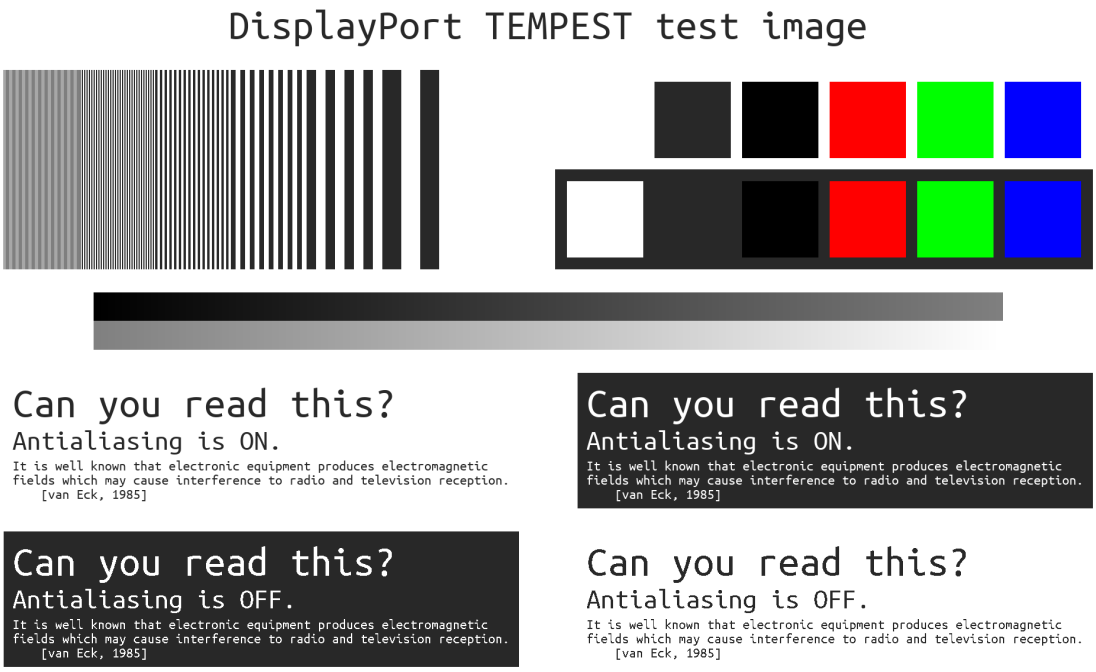


Figure 3.9: Test image.

since the average padded line length n_p is not an integer, the total length of a 512-line period can vary by one byte. Additionally, as described in Section 3.1.5, 512-line periods whose first line is in the vertical blanking period start six bytes earlier than what the DisplayPort standard specifies. Therefore, even with perfectly accurate $\mathbf{X}_\mathrm{o}$, this model cannot achieve a zero error. The fraction of inter-reset periods that are one byte longer depends on the video mode, which causes the average error to depend on the mode.

3.5.3 Test image

Figure 3.9 shows a test image to demonstrate the behaviour of this attack. It contains mainly six colours: white background (`#ffffff`), dark gray foreground (`#282828`), black (`#000000`), red (`#ff0000`), green (`#00ff00`) and blue (`#0000ff`), plus two gradient bars (and some gray edge pixels in anti-aliased text).

Figure 3.10 shows a two-channel grayscale image reconstructed from emissions recorded 2 m from the display that shows Figure 3.9. The brightness of pixels in this output is proportional to the difference $\bar{u}_2[i] - \bar{u}_1[i]$ between the foreground and background channels (silhouette mode). The gray areas on the left and bottom of Figure 3.10 are the horizontal and vertical blanking periods, respectively, and the vertical stripes are the fill bytes inserted as padding with each transfer unit.

In the reconstructed image, the `00` padding bytes appear more similar to the white background than the dark gray foreground, because the `00` bytes are a complement of the

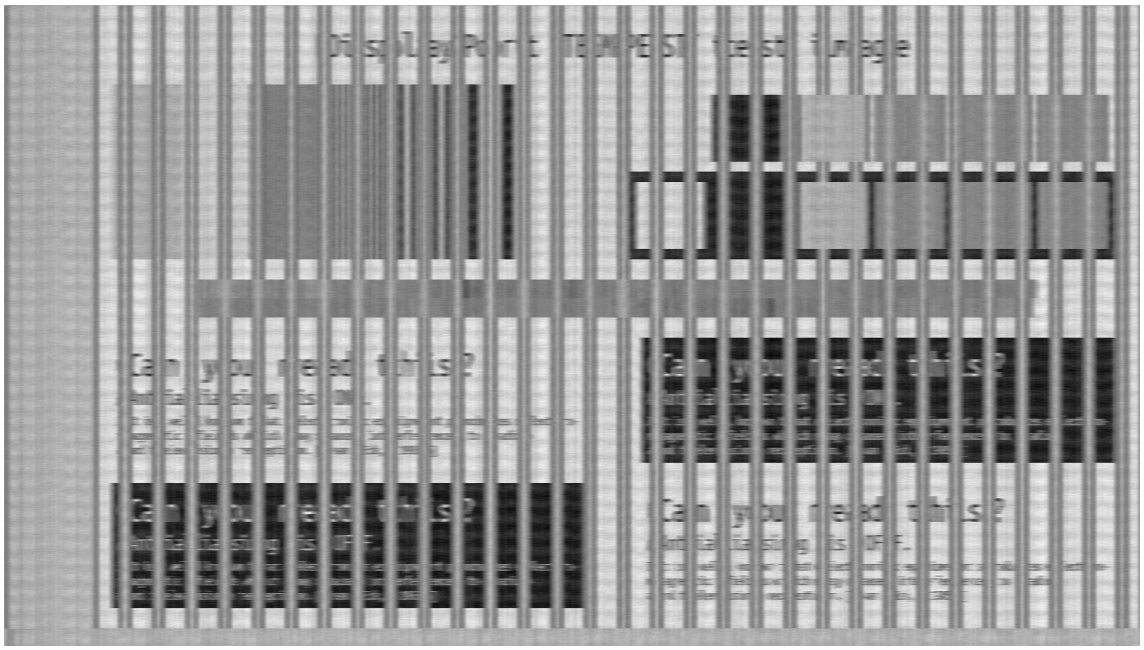


Figure 3.10: Silhouette mode reconstructed test image from Figure 3.9, showing the difference between the foreground (dark gray) and background (white) channels. The image was not yet rescaled or processed to remove fill and blanking periods. The image was reconstructed from 10 recordings, each 2 s long, made at 2 m antenna distance.

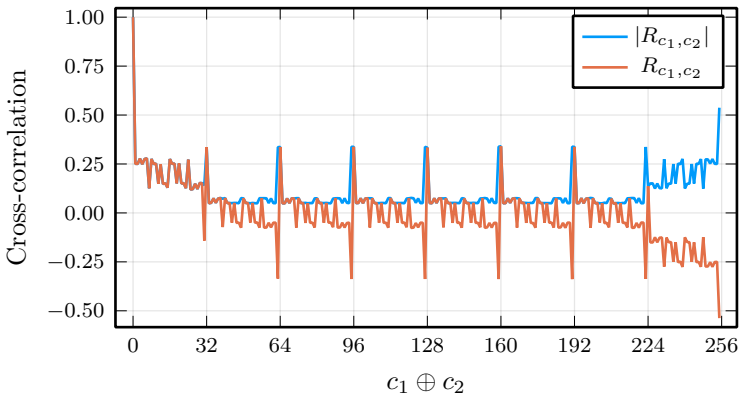
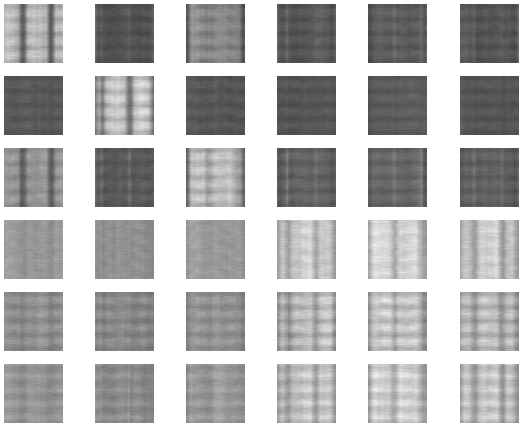


Figure 3.11: Cross-correlation R_{c_1,c_2} of scrambled and encoded data for two constant-value streams with byte values c_1 and c_2, as a function of their exclusive-or $c_1 \oplus c_2$.

Table 3.3: Samples of reconstructed channels corresponding to each colour in the test image (rows), showing the output for each coloured square (columns).

Channel	Colour in displayed image					
	#FFFFFF	#282828	#000000	#FF0000	#00FF00	#0000FF
						

`ff` bytes of the background, and therefore the respective scrambled byte sequences will also be complements. As explained in Section 3.1.4, inverting a byte often also inverts the corresponding 10-bit symbol. Therefore, the bitstreams transmitted for a byte and its complement are negatively correlated, which results in similar output because we look only at the absolute value of the correlation.

Similarly, the bitstreams corresponding to colours that match or complement each other in either encoder block (5-bit or 3-bit) will be correlated, as can be seen in Figure 3.11. As a result, a byte value and its complement are the most difficult pair of values to distinguish from the magnitude of the correlation.

Table 3.3 shows in six rows the six channels $\bar{u}_1, \ldots, \bar{u}_6$ corresponding to the six main colours used in the test image, and shows in six columns the respective section of the test image that contains that colour. This demonstrates a limitation of this attack: it is not able to distinguish pure red, green, and blue areas from each other, because the corresponding pixel values are cyclic shifts of the same three-byte sequence. (This limitation does *not* apply to colour combinations where the corresponding RGB byte sequences are *not* exact cyclic shifts of each other, as we will see later in Section 3.5.5.)

This limitation exists because the templates I build are not always aligned with the same colour component (red, green, or blue) in the datastream. If the datastream only contained pixel data, a template corresponding to the byte sequence `ff 00 00 ff 00 00` ... would always match one of the three colours, with `ff` always aligned with the same component. However, since the blanking periods and fill regions are not an integer number of pixels long (i.e. their lengths in bytes are not divisible by 3), such a template can match one

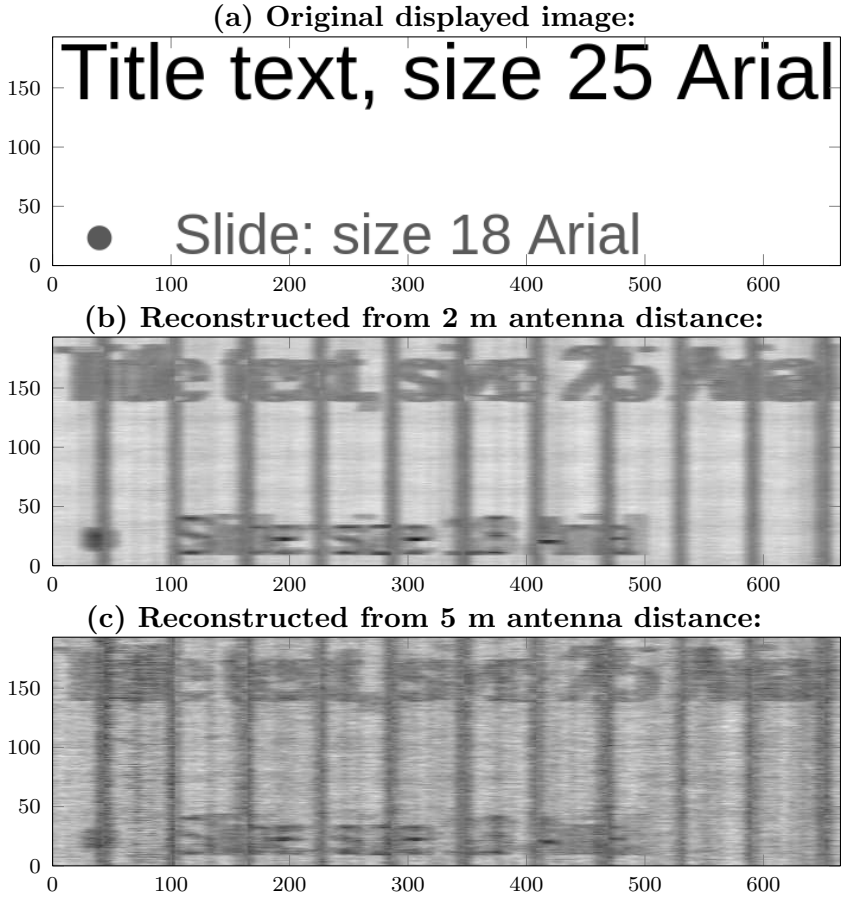


Figure 3.12: Cropped original and silhouette mode reconstructed images, showing the channel difference $\bar{u}_2[i] - \bar{u}_1[i]$ between the foreground (gray) and background (white) channels. Channel scores were periodically averaged over 10 recordings, each 2 s long.

colour before a fill region, but another afterwards.

A more accurate model of the scrambler resets is necessary to avoid this issue. When constructing the templates, one would have to incorporate the predicted positions of the fill regions to calculate which byte corresponds to which component. Both accurate reset tracking and fill-aware templates will be discussed in Chapter 4.

3.5.4 Slideshow

As a more realistic scenario, I tested the attack with the monitor showing a fullscreen Google Slides presentation, with the default theme and settings. A cropped sample of the slide used is shown in Figure 3.12(a).

The slide mainly uses three colours: white (`#ffffff`) background, black (`#000000`) title text and gray (`#595959`) body text. Due to anti-aliasing, some other colours also appear at character edges.

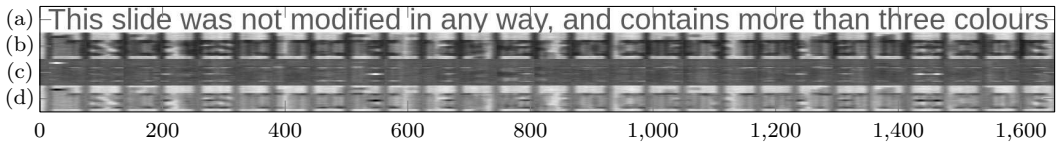


Figure 3.13: Displayed Google Slides text (a), background (b) and foreground (c) channels reconstructed from emissions recorded at a distance of 2 m, and the final channel-difference image (d).

Cropped samples of the reconstructed images, using emissions recorded at 2 m and 5 m distance, are shown in Figures 3.12(b) and 3.12(c). I rescaled the image horizontally by a factor 4:3 (4 lanes, 3 bytes per pixel) to match the monitor's aspect ratio, and removed columns containing fill bytes. Since the sliding-window width w contains fill bytes when centred on an image byte near a fill region, narrow vertical stripes are still present in the output.

Figure 3.13 shows another zoomed-in line of slide text, the foreground and background channels computed from emissions captured 2 m away from the monitor, and the reconstructed image. Due to antialiasing, only a small number of consecutive gray pixels with the foreground colour appear in the displayed image. Because of this, the reconstructed foreground channel contains little information about the text other than horizontal lines, such as in those in "T" and "e". The antialiased text, however, is still readable in the background channel, which lets the eavesdropper distinguish the background from all other pixel colours.

3.5.5 Colours

To demonstrate that the attack is capable of distinguishing more than just a single colour pair, I use the sixteen-colour test image shown in Figure 3.14, which features characters from the *Noto Emoji* font. I suppressed antialiasing to ensure that the image contains exactly sixteen different colours. As described in Section 3.4, I produce a multi-coloured output image by colouring each pixel according to the channel $\operatorname{argmax}_k \bar{u}_k$ with the highest score at that position. The raw reconstructed image, without rescaling or removing fill regions, is shown in Figure 3.15.

Large single-colour areas are reconstructed correctly, with only occasional errors. Most misclassified pixels are near boundaries between colours or adjacent to fill regions, where the sliding window used to compute correlations covers different-coloured pixels.

Figure 3.16 shows the same reconstruction with fill byte regions removed and the rest of the image rescaled accordingly. Narrow vertical stripes remain at image pixels where the correlation window partially overlaps fill regions.

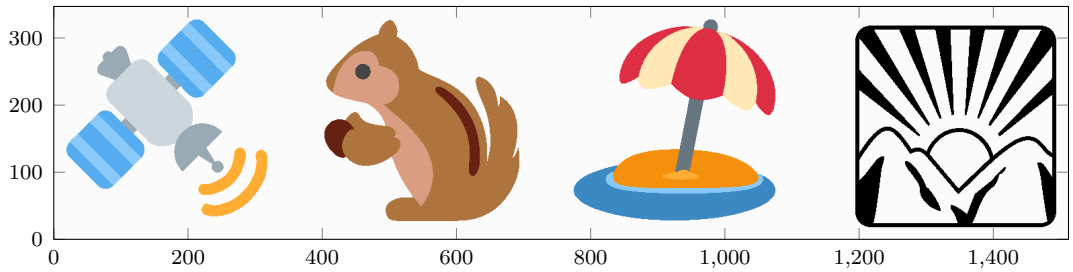


Figure 3.14: Sixteen-colour test image.

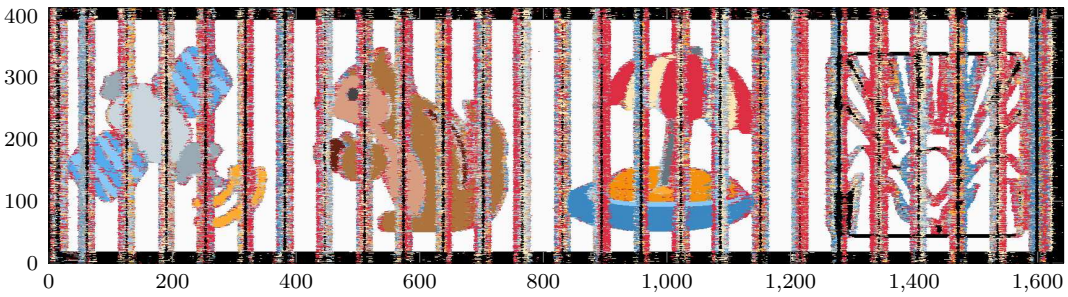


Figure 3.15: Image from Figure 3.14 reconstructed from 2 m antenna distance, with pixel colours chosen using voting mode, before post-processing.

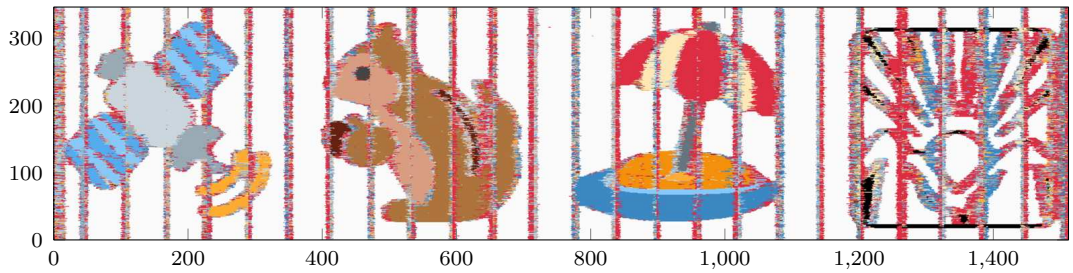


Figure 3.16: Image from Figure 3.14 reconstructed from 2 m antenna distance, after removing fill regions and rescaling horizontally to match the original aspect ratio.

Chapter 4

Accurate scrambler tracking for colour enumeration

4.1 Introduction

In the previous chapter, I demonstrated that an eavesdropper can reconstruct the image shown on a monitor from several metres away, by exploiting electromagnetic emissions which are unintentionally radiated from a DisplayPort cable carrying the image. I described a synchronisation algorithm that identifies times at which the DisplayPort scrambler is in its initial state, allowing the eavesdropper to synthesise a local replica of the scrambler byte sequence. I then assumed that the eavesdropper is interested in a small set of colours that they know are present in the image (e.g. common slideshow or document theme colours, or those used in specialised software such as the user interface of a voting machine). For each colour, my second algorithm creates a template of the expected emitted signal, based on the scrambler replica, and reconstructs the image from short-term correlations between the templates and the received signal.

In this chapter, I will extend these two algorithms to address their shortcomings. First, I describe a synchronisation algorithm that uses a tracking loop to significantly reduce the synchronisation error. I then introduce an efficient algorithm for colour enumeration, which allows the eavesdropper to express template correlations for all 2^{24} RGB colours using only 61 dot products, and from these efficiently extract a set of most likely colours for an image region.

The previous synchronisation algorithm describes scrambler timing with three parameters: a reset period R between consecutive "scrambler reset" symbols, which are inserted every 512 lines, an overflow period P corresponding to the $2^{16} - 1$ byte period of the scrambling sequence, and a time offset x_0. I demonstrated that this simple model cannot fully describe the reset times, i.e. that its error (difference between observed and predicted reset times) will be non-zero even if the input is completely accurate. For low-noise recordings, the

average error was between 0.5 and 1.7 samples, or between 1.6 and 5.4 transmitted bytes, depending on the target display video mode. This error cannot be eliminated by improving the signal-to-noise ratio or with better algorithms using the same reset model, since it is caused by a simplifying assumption that the reset period R is constant. In practice, the length of a video line in a DisplayPort data stream can vary by a byte, due to what we might call *leap bytes*, which are inserted at the end of some lines to meet the target average padded length. Since the number of such leap bytes will not be the same in every 512-line interval between scrambler resets, the reset period varies.

The second source of inaccuracy is the assumption that the bitrate of the DisplayPort link is exactly equal to the nominal 1.62 GHz specified by the standard. Due to clock inaccuracy (caused both by manufacturing tolerance and thermal drift), this means that the local scrambler replica's timing slowly drifts out of sync with the actual scrambler. Frequent resynchronisations after each overflow period P limit the impact of this drift on the previous algorithm, since the error only accumulates within a single period, but the total timing error for a scrambler byte in the middle of the stream is still greater than the reset model's error.

The first algorithm presented in this chapter addresses the latter by treating the bitrate as time-variable. The algorithm maintains an estimate of the bitrate $f_\mathrm{b}(t)$ by measuring the offset between tracked and real scrambler positions every line and updating the estimate using a tracking loop. I do not try to eliminate the error in scrambler reset times, and still use the same algorithm that assumes a constant period R. Leap bytes therefore still contribute to the synchronisation error, but the tracking loop brings the error back close to zero quickly. I chose this approach because the DisplayPort standard does not fully specify how the data is padded, and in my reverse engineering attempts, while I determined how fill bytes are inserted (Algorithm 1), I did not yet manage to fully deterministically model the algorithm determining which lines contain an additional leap byte. Additionally, since the padding details are implementation-defined, a model relying on a reverse-engineered leap byte algorithm might only be applicable to the specific video display controller I analysed.

Section 4.2 describes the architecture of a scrambler tracking delay-locked loop based on a *proportional-integral-derivative (PID) controller*, and two discriminators I use to measure the tracking error: a slow search for a cross-correlation maximum, used during acquisition (before the controller locks on, i.e. while the error is still large), and a fast estimate similar to the *early power minus late power* discriminator used in GPS receivers [73]. I also describe an efficient implementation of arbitrary-offset sampling used by the discriminators.

Next, Section 4.3 describes the properties of the 8b/10b encoder used in DisplayPort which allow me to significantly reduce the computational cost of correlating the received signal with templates for all 2^{24} RGB colours. I describe an algorithm that takes advantage of these properties to express the correlation between the received signal and a colour

template as a sum of seven correlations with sub-templates, and only requires 61 such sub-template correlations which fully describe the result for all 2^{24} colours. Using this optimisation, my algorithm can return a sorted list of the 1000 highest-correlation colours for an image line in 4 ms.

4.2 Scrambler code phase tracking

To reconstruct image information from a signal $s[i]$ containing DisplayPort emissions, the eavesdropper needs to construct a local replica of the scrambler, phase-aligned with the received signal. This is similar to the *code tracking* portion of a GPS receiver [74], which aligns the received signal with a replica of the pseudorandom code transmitted by a satellite.

We can represent this alignment as an array $i_\mathrm{s}[i]$ of scrambler byte positions (i.e. *code phase*) corresponding to each signal sample, meaning that at the time when sample $s[i]$ was taken, the DisplayPort LFSR produced the $i_\mathrm{s}[i]$-th byte of the (periodic) scrambling sequence. As the sample can be taken between two bytes, these positions are real-valued.

The synchronisation algorithm presented in this chapter builds upon the reset model described in Chapter 3. I first search for a subset of times at which the scrambler resets or overflows, and from these predict a set of reset times $\mathbf{X}$. Our previous scrambler alignment algorithm would then assume that the DisplayPort bitrate is constant and equal to the nominal $f_\mathrm{b} = 1.62$ GHz (or, depending on the video mode, a higher bitrate from the DisplayPort standard). The scrambler positions would then be

$$i_\mathrm{s}[i] = \begin{cases} 0, & \text{if } i \in \mathbf{X} \\ i_\mathrm{s}[i-1] + f_\mathrm{b}/(10 f_\mathrm{s}), & \text{otherwise} \end{cases} \tag{4.1}$$

The factor 10 here converts from bit positions to byte positions, since each 8b/10b-encoded byte is 10 bits long. Note that for simplicity, this equation assumes that all resets in $\mathbf{X}$ happen at integer sample positions.

Assuming that the bitrate is exactly f_b limits the accuracy that can be achieved by this model. In practice, the bitrate is not exactly equal to the nominal rate, since the oscillator that clocks the DisplayPort signal is not perfectly accurate. The maximum in-spec difference between the nominal and actual frequency is stated by the oscillator manufacturer as its *frequency tolerance*. The frequency will also vary over time, mainly due to changes in temperature, changes in crystal structure, and ageing, up to the oscillator's *frequency stability*. For common crystal oscillators, both frequency tolerance and frequency stability are in the 0–50 ppm (parts per million) range.

The clock rate of the SDR receiver, and therefore also the sampling rate f_s, can similarly vary due to oscillator frequency error. For the USRP X300 I used, the specified reference

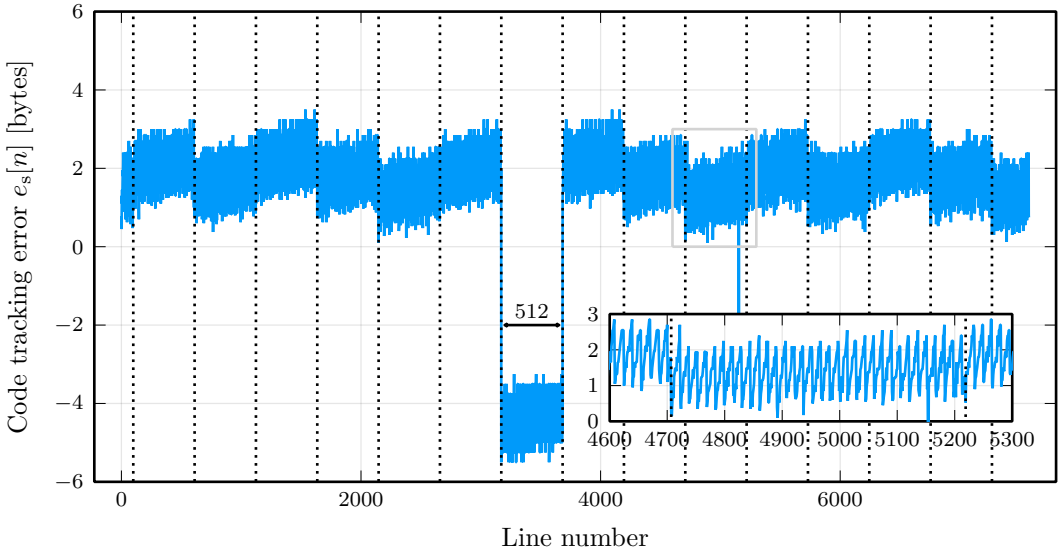


Figure 4.1: Scrambler tracking error when only using the reset model (Equation 4.1), measured once per line as the position of the cross-correlation maximum $e_s[i]$, for a recording made at 2 m antenna distance.

clock accuracy is 2.5 ppm [75]. The algorithms described in this thesis only use the ratio f_b/f_s, rather than the individual values, and I will consider f_s to be exact and absorb any errors into the tracked bitrate $f_b(t)$.

We can roughly estimate the actual bitrate from the reset model's overflow period P. The scrambler overflows every $2^{16} - 1$ bytes, meaning that $P = (2^{16} - 1)\frac{f_b}{10f_s}$. In my experiments, after all devices warmed up for a few minutes to minimise thermal drift, the bitrate estimate computed from P differed from the nominal value by around 30 ppm.

Due to the mismatch between the nominal and real bitrate, the estimated scrambler position $i_s[i]$ will drift over time from the actual position. Resetting the scrambler replica (at times in $\mathbf{X}$), also resets the accumulated position drift, which keeps the maximal code phase error due to bitrate mismatch bounded. This happens every $2^{16} - 1$ bytes, or P samples, at the overflow period of the LFSR. Assuming that the relative frequency error is $\delta_f = 30$ ppm (i.e. the real bitrate is $(1 + \delta_f)f_b$), the accumulated code phase error over that time period will be

$$\delta_f \cdot (2^{16} - 1) \approx 1.97 \text{ bytes}. \tag{4.2}$$

Figure 4.1 shows a plot of the code phase error of the output of this algorithm (Equation 4.1), measured in the horizontal blanking region of each line using an approach described later in Section 4.2.2. The short "zig-zag" period is the scrambler overflow period P, after which the accumulated error is eliminated by resetting the scrambler position to zero, as predicted by the reset model. Every 512 lines, when the scrambler is reset, there is another change in the error due to the total length of the lines not being constant. The total number of bytes may differ by one between reset periods, since the number of lines

containing more fill bytes than average may vary by one in a 512-line period. Therefore, the total number of bytes between consecutive scrambler resets is either slightly lower than the average period R computed by the reset model (resulting in increased error), or slightly higher (decreased error).

In the 512-line period from line 3171 to line 3682, the code phase error is shifted by six bytes compared to the rest of the recording. I believe that this is due to a bug in the display controller described in Section 3.1.5: in the vertical blanking region, blanking start symbols are inserted six bytes earlier than they would be in displayed image lines.

Scrambler alignment only using a reset model is insufficient for byte-level alignment, but still tolerable for image reconstruction: since $s[i]$ is band-limited, the expected autocorrelation of received emissions is still non-zero with a two-byte lag. Additionally, the 1.97 sample drift between resets estimated in Equation 4.2 is an upper bound, and the actual drift will be lower for samples that are well before an overflow period. More complex image reconstruction algorithms that make use of the exact position of fill and blanking regions, such as the colour enumeration algorithm in Section 4.3, however, benefit from more accurate synchronisation.

In the rest of this section, I present an algorithm that aligns the scrambler with $s[i]$ more accurately than the simple nominal-frequency model. I start with a short high-level overview, and then discuss the details of code phase discriminators, the tracking loop used to estimate the bitrate, and my approach for choosing its controller parameters.

4.2.1 Overview

The tracking algorithm is based on a delay-locked loop, which maintains an estimate $f_\mathrm{b}[i]$ of the bitrate. This loop consists of two components: an estimator for the difference between the output position $i_\mathrm{s}[i]$ and the true scrambler position (i.e. code phase error), and a PID controller which updates the bitrate based on this error. This update is made once per line, in the horizontal blanking region, since I use the known zero bytes in the blanking region to measure the error.

The scrambler positions are then calculated similarly to Equation 4.2:

$$i_\mathrm{s}[i] = \begin{cases} 0, & \text{if } i \in \mathbf{X} \\ i_\mathrm{s}[i-1] + 10 f_\mathrm{b}[i]/f_\mathrm{s}, & \text{otherwise.} \end{cases} \tag{4.3}$$

Figure 4.2 illustrates the main signal processing components of the tracking algorithm.

4.2.2 Discriminators

The algorithm uses two *phase discriminators* to estimate the code phase error $e[i]$, both of which are based on the cross-correlation between $s[i]$ and templates for the expected signal

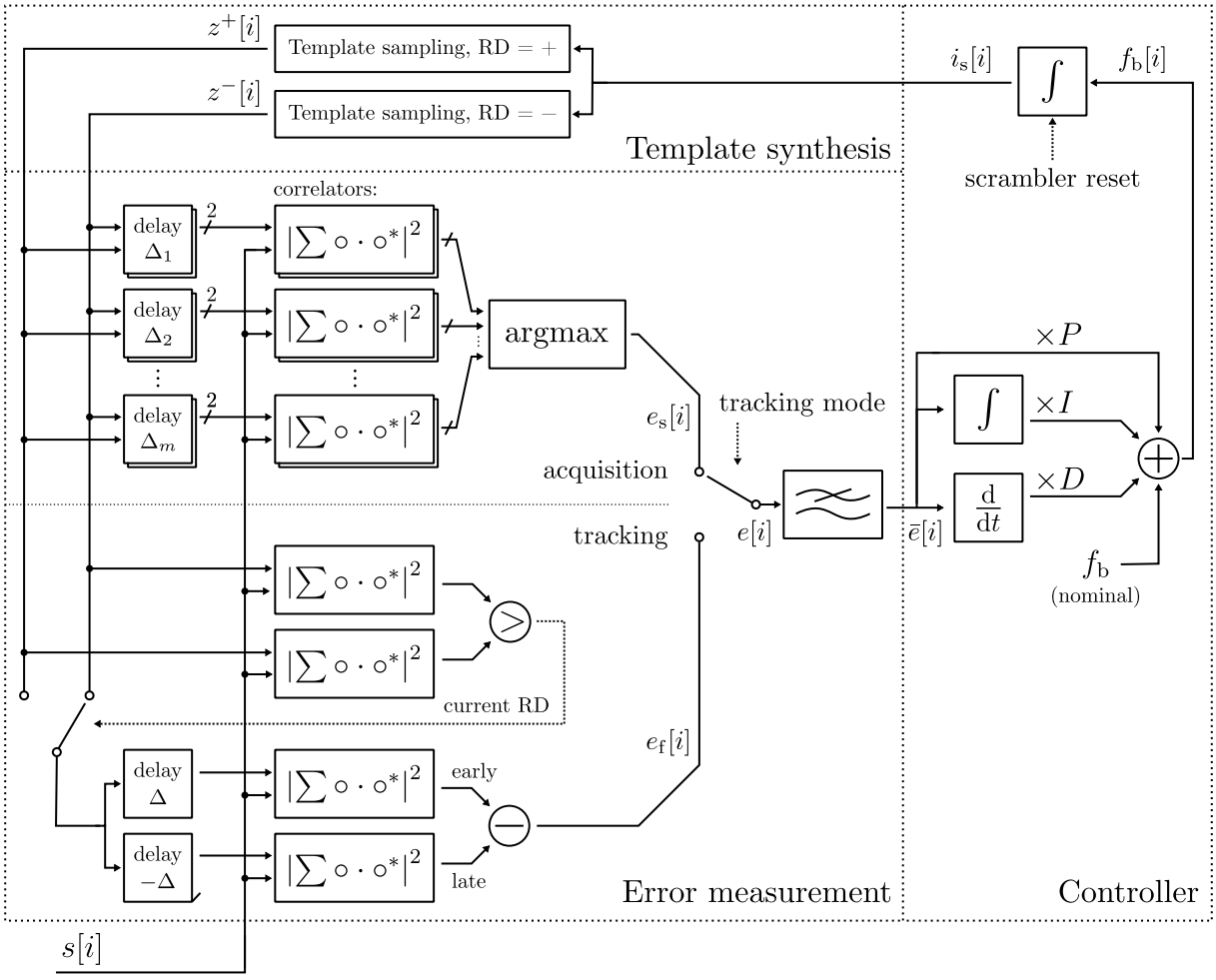


Figure 4.2: Overview of the scrambler tracking algorithm.

in the blanking region. The first, e_s, searches for the scrambler position near $i_\mathrm{s}[i]$ that maximises that cross-correlation. This search is slow, but it computes the offset directly and is correct for large offsets, which makes it suitable in the *acquisition* phase, where the tracking loop is not yet tracking the scrambler closely. The second discriminator, e_f, is an *early power minus late power* discriminator, similar to that used in a noncoherent GPS delay-locked loop. This is significantly cheaper to compute, since the discriminator only evaluates the cross-correlation at four offsets, but it is only a reliable estimate if the code phase error is low (in the *tracking* phase).

The algorithm starts in acquisition mode, where $e[i] \coloneqq e_\mathrm{s}[i]$, and stays there for the first 200 lines to give the tracking loop time to lock onto the scrambler. Afterwards, it switches to tracking mode, where $e[i] \coloneqq e_\mathrm{f}[i]$, to reduce computational load. Every 512 lines, when the scrambler resets, the offset between the received signal and our scrambler replica increases if scrambler reset byte is not exactly at the position in $\mathbf{X}$ predicted by the reset model. At these resets, the algorithm briefly switches back to acquisition mode for ten lines to give the tracking loop time to bring the error down to the range where e_f-based tracking can work.

The templates used by the discriminators are the expected emissions from a scrambled, 8b/10b encoded DisplayPort data stream containing only zero bytes, as it would be received by the eavesdropper (i.e. downconverted and lowpass filtered by an antialiasing filter) during a blanking period. Because the 8b/10b encoder has one bit of internal state, the *running disparity*, the emissions in a blanking region will match one of the two templates: $z^+(i)$, with positive initial running disparity, and $z^-(i)$, with negative initial disparity.

I treat the templates as continuous functions that map a real-valued scrambler byte position i to $z^+(i)$ and $z^-(i)$, the downsampled and filtered signals sampled at time $i/(f_\mathrm{b}/10)$. I will discuss the details of template construction and an efficient implementation of such arbitrary-offset sampling later in Section 4.2.6.

Let n_b be the length of the horizontal blanking region in samples, computed from the sampling rate f_s and the target display's video mode displayed width w_d, full width (including the blanking region) w_t, and pixel rate f_p. The value is rounded to the nearest sample, since we will use it as the length of the correlation window:

$$n_\mathrm{b} = \left\lfloor \frac{(w_\mathrm{t} - w_\mathrm{d}) f_\mathrm{s}}{f_\mathrm{p}} + 0.5 \right\rfloor . \tag{4.4}$$

Both discriminators are based on cross-correlations $R^+(i,\delta)$ and $R^-(i,\delta)$ between $s[i]$ and the templates, starting at signal sample i and the corresponding scrambler position $i_\mathrm{s}[i] + \delta$. They are computed over the length of the blanking region (i.e. from the current index i to $i + n_\mathrm{b} - 1$):

$$R^+(i,\delta) = \sum_{0 \le d < n_\mathrm{b}} s[i+d] z^+(i_\mathrm{s}[i] + 10 d f_\mathrm{b}/f_\mathrm{s} + 10\delta)^* \tag{4.5}$$

with R^- defined analogously as cross-correlation with z^-. Figure 4.3 shows the expected shape of R^+ and R^-, and parameters used for the discriminators.

For both discriminators, I first evaluate $R^+(i,0)$ and $R^-(i,0)$, at the zero offset, to select the template best matching the current encoder running disparity. For the rest of this section, I will assume (without loss of generality) that $|R^+(i,0)| > |R^-(i,0)|$, and therefore that the encoder state matches z^+; if $|R^+(i,0)| < |R^-(i,0)|$, the only change is in the choice of template.

The slow discriminator e_s then simply samples the cross-correlation at a range of offsets $\delta \in \mathbf{D}$ and computes the phase offset as the position of the maximum. The offsets in $\mathbf{D}$ range from -6 to 6, with dense sampling near zero in steps of 0.025, and coarser steps farther away.

$$e_\mathrm{s}[i] = \operatorname*{argmax}_{\delta \in \mathbf{D}} |R^+(i,\delta)| \tag{4.6}$$

The fast discriminator e_f is an *early power minus late power* discriminator. It computes the difference between the squared magnitude of outputs of an early and a late correlator,

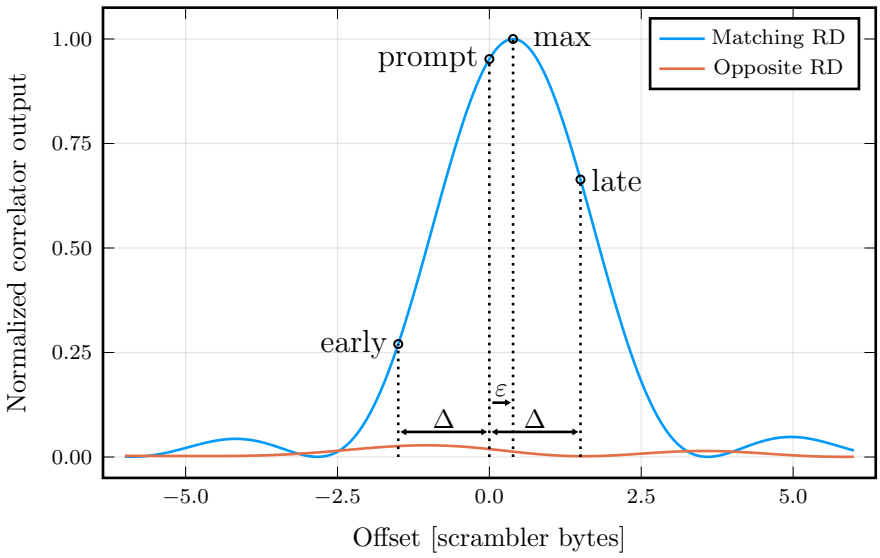


Figure 4.3: Expected correlation $R^+(i, \delta)$ and $R^-(i, \delta)$, for varying offset δ, showing points of interest for the code phase discriminators.

which are positioned at $i_s[i] \pm \Delta$. The difference is normalised by the prompt power (at the zero offset) to reduce dependence on signal power.

$$e_f[i] = \frac{|R^+(i, -\Delta)|^2 - |R^+(i, \Delta)|^2}{|R^+(i, 0)|^2} \tag{4.7}$$

The discriminator output is only meaningful if at least one of the correlators is on the main slope of the correlation shown in Figure 4.3. If the current code phase error is small, and so the early and late correlator are on opposite sides of the slope near the peak, it is well-estimated by the discriminator. As this error increases, the sign of $e_f[i]$ stays correct until both correlators are no longer on the slope, but the magnitude is no longer accurate.

Choosing a large value of Δ would make the discriminator more robust, but would also make it a worse estimate of the true code phase error, since the approximation $e_f[i] \approx e_s[i]$ is closer for small values of Δ. I set $\Delta = 1.5$ bytes, for which, as shown in Figure 4.4, the discriminator is linear if the error is under 0.5 bytes for the receiver bandwidth I used.

4.2.3 Tracking loop

The tracking controller is updated every line, in the horizontal blanking region. We can compute the positions in the signal $s[i]$ at which the blanking region begins from the reset model. Since the scrambler resets at the beginning of a blanking region, one blanking region begins at x_0. The reset period is 512 lines, and so the average length of a line is $n_p = R/512$, and the line rate $f_h = f_s/n_p = 512 f_s/R$. Updates, therefore, happen at indices $i \in \mathbf{B}$, where

$$\mathbf{B} = \{\lfloor x_0 + k n_p + 0.5 \rfloor : k \in \mathbb{Z}\} \tag{4.8}$$

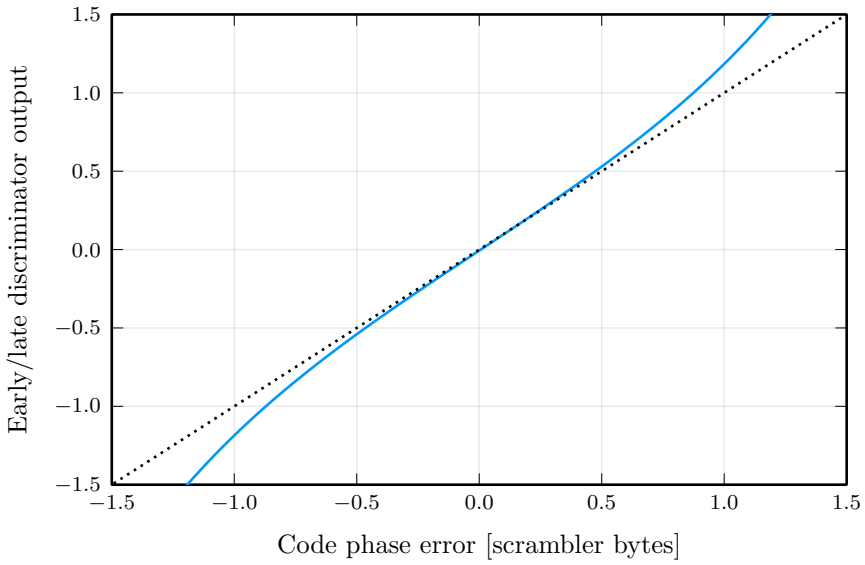


Figure 4.4: Plot of early power minus late power discriminator output as the function of the true code phase offset. Dotted line $y = x$ added as a guideline.

Between updates, the tracked bitrate $f_\mathrm{b}[i]$ stays constant.

Let $i \in \mathbf{B}$ be the current index in $s[i]$, and k the corresponding line number. The input to the PID controller is an exponential moving average $\bar{e}[k]$ of the code phase error estimate $e[i]$, with $\alpha = 0.8$:

$$\bar{e}[k] = \alpha\bar{e}[k-1] + (1-\alpha)e[i] \tag{4.9}$$

The averaging is done to reduce the impact of outliers due to noise or large values $e[i]$, so that the controller adjusts $f_\mathrm{b}[i]$ gradually instead of a large change in a single update.

The smoothed offset is then used as the input to a PID controller with coefficients $(K_\mathrm{P}, K_\mathrm{I}, K_\mathrm{D})$, which outputs a frequency correction $\Delta f_\mathrm{b}[i]$: the difference between the nominal bitrate and the current estimate. A typical continuous-time PID controller would compute the correction as

$$\Delta f_\mathrm{b}(t) = K_\mathrm{P}\bar{e}(t) + K_\mathrm{I}\int_{t'\le t}\bar{e}(t')\mathrm{d}t + K_\mathrm{D}\frac{\mathrm{d}\bar{e}(t)}{\mathrm{d}t} \tag{4.10}$$

which I discretise as

$$\Delta f_\mathrm{b}[k] = K_\mathrm{P}\bar{e}[k] + K_\mathrm{I}f_\mathrm{h}^{-1}\sum_{m\le k}\bar{e}[m] + K_\mathrm{D}f_\mathrm{h}(\bar{e}[k] - \bar{e}[k-1]) \tag{4.11}$$

by approximating the integral as a sum of samples multiplied by the controller update period f_h^{-1}, corresponding to the video line rate f_h, and the derivative as a similarly scaled difference.

The output of the tracking loop is then simply $f_\mathrm{b}[i] = f_\mathrm{b} + \Delta f_\mathrm{b}[k]$.

4.2.4 Parameter choice

I chose the PID controller parameters using a "black-box optimiser" provided by the BlackBoxOptim.jl Julia package [76], which uses an adaptive differential evolution optimiser. I made ten recordings of DisplayPort emissions at a short distance, approximately 1 m, with the screen showing a single-colour image for a randomly chosen colour. The goal function minimised by the optimiser was the average of $|e_\mathrm{f}[i]|$ over the first three frames in each of the recordings, with 10% of the largest and smallest values removed to ignore the lines in which the controller did not yet lock onto the scrambler.

The optimiser was used to find the sampling rate dependent values $(K_\mathrm{P}, K_\mathrm{I} f_\mathrm{h}^{-1}, K_\mathrm{D} f_\mathrm{h})$ from Equation 4.11. The search range for all three values was $[0, 2\ \mathrm{MHz/byte}]$, chosen to be a few times larger than the highest values that for which tracking was successful in my initial (manual) tuning experiments.

The optimiser converged after 43 steps (≈ 2.5 hours on a single Intel i5-6300U core). The best-performing coefficients found were

$$(K_\mathrm{P}, K_\mathrm{I} f_\mathrm{h}^{-1}, K_\mathrm{D} f_\mathrm{h}) = (446.2, 46.6, 584.9)\ \mathrm{kHz/byte}. \tag{4.12}$$

The corresponding rate-independent coefficients, substituting $f_\mathrm{h} = 37.88$ kHz, are

$$(K_\mathrm{P}, K_\mathrm{I}, K_\mathrm{D}) = (446.2\ \mathrm{kHz/byte}, 1767\ \mathrm{kHz^2/byte}, 15.44\ \mathrm{byte^{-1}}). \tag{4.13}$$

While these parameters were optimised using only short-distance recordings, scrambler tracking in later experiments was also successful at longer distances. As we will see below, they result in a wide loop bandwidth, and therefore fast settling time after scrambler resets.

To calculate the loop bandwidth, let $i_\mathrm{s}[k]$ be the DLL-tracked code phase at line k, and $i_\mathrm{r}[k]$ the true code phase of the incoming signal. After the loop locks onto the signal and $i_\mathrm{s}[k] - i_\mathrm{r}[k]$ is small, we can approximate the discriminator output as a true measurement of the phase error:

$$e[k] = i_\mathrm{r}[k] - i_\mathrm{s}[k]. \tag{4.14}$$

Let $X(z)$, $Y(z)$, and $E(z)$ be the z-transforms of $i_\mathrm{r}[k]$, $i_\mathrm{s}[k]$, and $e[k]$ respectively, and $G(z)$ the z-transform of the open-loop transfer function of the DLL, where $Y(z) = G(z)E(z)$ (i.e. $G(z)$ jointly describes the lowpass filter, PID controller, and integrator). The corresponding closed-loop transfer function $H(z)$ is then

$$H(z) = \frac{Y(z)}{X(z)} = \frac{G(z)}{1 + G(z)}. \tag{4.15}$$

We can write $G(z) = G_1(z)G_2(z)G_3(z)$ as the product of transfer functions for each of the three stages: the lowpass filter

$$G_1(z) = \frac{1 - \alpha}{1 - \alpha z^{-1}}, \tag{4.16}$$

the PID controller

$$G_2(z) = K_\mathrm{P} + K_\mathrm{I} f_\mathrm{h}^{-1} \frac{1}{1 - z^{-1}} + K_\mathrm{D} f_\mathrm{h} (1 - z^{-1}), \tag{4.17}$$

and the integrator which computes the current code phase (note that here we also treat $i_\mathrm{s}[k]$ as sampled at the line rate)

$$G_3(z) = \frac{1}{10} f_\mathrm{h}^{-1} \frac{z^{-1}}{1 - z^{-1}} \tag{4.18}$$

where the 1/10 factor converts from the tracked bitrate to bytes, since discriminators measure the phase error in bytes.

Substituting all into Equation 4.15 lets us compute $H(z)$, and from it the 3 dB loop bandwidth $f_{3\ \mathrm{dB}}$ by numerically solving

$$\left| H(\mathrm{e}^{2\pi f_{3\ \mathrm{dB}}/f_\mathrm{h}}) \right| = \frac{1}{\sqrt{2}} \tag{4.19}$$

which results in $f_{3\ \mathrm{dB}} \approx 6.45$ kHz, i.e. approximately 17% of the line rate (and therefore the controller update rate), or $4 \cdot 10^{-6} \cdot f_\mathrm{b}$ in terms of the DisplayPort bitrate.

This wide loop bandwidth results in a short impulse response, at the cost of relatively little phase noise attenuation. A short impulse response is desirable for scrambler tracking, as it ensures that the loop settles quickly after phase jumps happening at scrambler resets.

4.2.5 Performance

Figure 4.5 shows the tracking error, measured using e_s, at the end of each line of a sample recording made at 2 m antenna distance. The recording is the same as the one used in the no-tracking Figure 4.1, which is also shown in the plot.

Table 4.1 shows the mean and standard deviation of the absolute value of tracking error $|e_\mathrm{s}[i]|$, averaged over five recordings in the colour enumeration dataset (Section 4.4.3) for each antenna distance.

Table 4.1: Code tracking error (measured as $|e_\mathrm{s}[i]|$) mean and standard deviation, averaged over five recordings for each antenna distance.

Distance	Mean (bytes)	Standard deviation (bytes)
2 m	0.176	0.1646
3 m	0.178	0.1564
4 m	0.224	0.2032

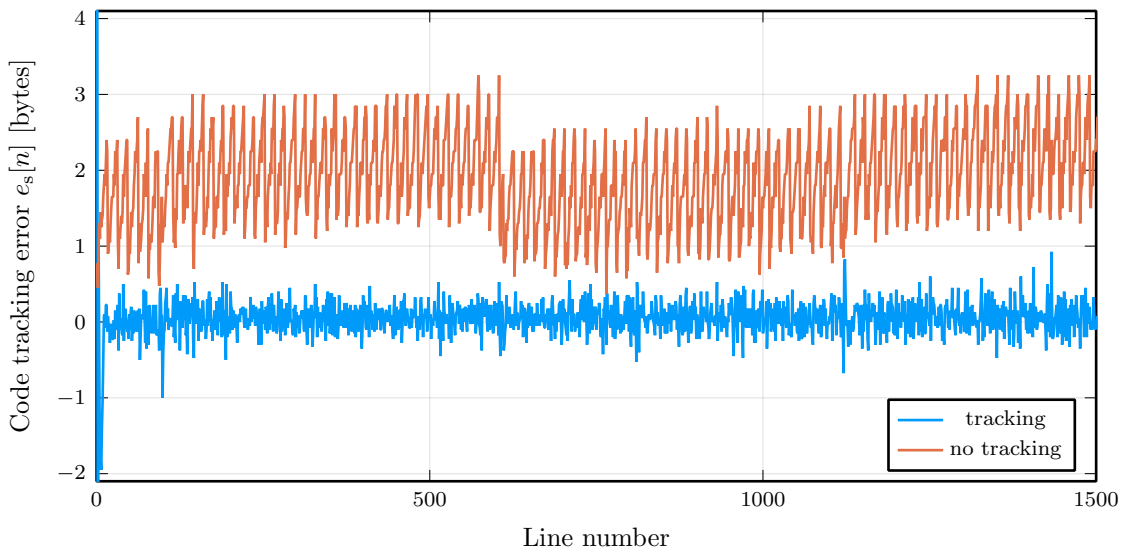


Figure 4.5: Code tracking error of PID-based scrambler tracking output, compared to alignment only using a reset model. The offset is measured as $e_s[i]$ once per line, for a recording made at 2 m antenna distance.

4.2.6 Implementation

The code phase discriminators require us to be able to efficiently sample templates $z^+(i)$ and $z^-(i)$ at arbitrary real-valued indices i. I will first show how $z^+(i)$ is constructed ($z^-(i)$ is identical, up to the initial running disparity), and then describe an efficient implementation using the Farrow filter structure, which is further detailed in Appendix B.

The template $z^+(i)$ contains the IQ-downconverted emissions resulting from a scrambled, 8b/10b-encoded DisplayPort lane transferring only zero-valued bytes. Since the template should match the output of our software-defined radio receiver, the emissions are down-converted by the tuning frequency f_c and lowpass filtered with cutoff frequency $B/2$, where B is the bandwidth of the SDR anti-aliasing filter.

Let $\xi[n] \in \{-1, 1\}$ be the array of scrambler bits transmitted over the targeted DisplayPort lane if the data is zero-valued, computed by 8b/10b encoding the scrambler output. We can model the emissions as a sampled signal with rate f_b, and values $s_1[n] = \xi[n]$.

After downconverting, we have:

$$s_2[n] = s_1[n] e^{-2\pi \mathrm{j} n f_c / f_b} \tag{4.20}$$

We can reconstruct the continuous-time emissions by sinc interpolating $s_2[n]$. If we scale the sinc kernel width by $(B/2)/f_b$, the interpolation can also serve as the antialiasing lowpass filter (constant factors ignored for simplicity):

$$z^+(i) = \sum_n s_2[n] \operatorname{sinc}\left(\frac{i - n(B/2f_b)^{-1}}{(B/2f_b)^{-1}}\right) = \sum_n s_2[n] \operatorname{sinc}\left(iB/2f_b - n\right). \tag{4.21}$$

Directly computing $z^+(i)$ using sinc interpolation would require recomputing and convolving with the sinc kernel for every desired fractional offset i. I instead use a Farrow filter, which is an efficient implementation of sinc interpolation and arbitrary-offset sampling further described in Appendix B. It is enough to construct the filter for a single period of the scrambler, instead of interpolating it over the entire length of the input recording.

The scrambler output $\xi[n]$ is periodic with period $T = 10 \cdot (2^{16} - 1)$, i.e. $\xi[n + T] = \xi[n]$. After downconversion, the sampled template $s_2[n]$ is almost periodic, up to a constant phase shift each period:

$$s_2[n + T] = \mathrm{e}^{-2\pi \mathrm{j} T f_\mathrm{c}/f_\mathrm{b}} s_2[n] \tag{4.22}$$

Substituting into Equation 4.21, we can see that the same holds for the interpolated template:

$$z^+(i + T) = e^{-2\pi j T f_\mathrm{c}/f_\mathrm{b}} z^+(i) \tag{4.23}$$

We can therefore use the Farrow filter to compute $z^+(i)$ for $0 \leq i < T$, constructed from $s_2[n]$ in that range (and sufficient samples to the left and right for the sinc kernel). For other positions $i + kT$, we can use the Farrow filter output with the appropriate phase correction:

$$z^+(i + kT) = \Delta\varphi[k] z^+(i) \tag{4.24}$$

where $\Delta\varphi[k] = e^{-2\pi j k T f_\mathrm{c}/f_\mathrm{b}}$ is a precomputed table of corrections.

4.3 Colour enumeration

In this section, I describe an algorithm for efficient colour enumeration for DisplayPort eavesdropping. The core of the algorithm is an efficient representation of templates (i.e. expected signals) for each colour as a linear combination of sub-templates, such that only a small number of sub-templates is needed.

I use the algorithm to search for the likely background colour of the target image, since my previous experiments showed that due to antialiasing, the text information an eavesdropper would likely be interested in is almost entirely contained in the background channel, in which the text shows up as a negative. After identifying the background colour, the image can be reconstructed using the same methods I previously demonstrated, possibly aided by the tracking loop.

I start this section with an overview of the relevant properties of the 8b/10b encoder, and then describe how they lead to the sub-template representation and how I prune sub-template combinations to efficiently compute a list of likely colours.

4.3.1 8b/10b encoding

Before transmission, DisplayPort pixel values are scrambled and then encoded using an 8b/10b encoder. The encoder can be thought of as two separate sub-encoders: 5b/6b, which uses the five low bits in the input byte and outputs six bits, and 3b/4b, which uses the three high bits and outputs four bits. The encoder has a single bit of state, shared between the sub-encoders: the *running disparity*, which is positive if the output so far contains more ones than zeroes, and negative otherwise.

For both sub-encoders, an input value can either have a single *balanced* encoding, with the same number of zeroes and ones, or two alternate *unbalanced* encodings, chosen depending on the running disparity to maintain balance in the output.

Due to the running disparity, changing a byte can affect not only the corresponding output symbol, but also later output. For example, consider a stream of pixels, all with the same colour (r, g, b). If one of the channel values is changed, for example r, the parts of the byte stream corresponding to the other two channels can change as well: a change in the running disparity (e.g. if a balanced symbol is replaced with an unbalanced one) can propagate to the other channels and change the selected encoding for an unbalanced sub-symbol.

Therefore, because of unbalanced symbols, the encoded output cannot be described as three interleaved symbol streams, one for each channel, which are encoded independently. Since my optimisations rely on splitting the output into independently encoded sub-templates, I will restrict the templates to only use balanced symbols. Due to scrambling, template bytes can be treated as independently chosen random values, and so they will include the same expected number of bits belonging to balanced symbols: $\approx 54\%$ of the output.

Let us define a *balanced 8b/10b encoding* as an analogue of the 8b/10b encoder, which behaves the same way for balanced sub-symbols, and outputs "nothing" for unbalanced sub-symbols. I will represent the output as a series of three-valued symbols in $\{-1, 0, 1\}$, where -1 and 1 correspond to zeroes and ones in the balanced output, and 0 to "nothing", meaning that the output is unbalanced and thus depends on the running disparity. Let the encoding function be

$$e : \mathbb{Z}_{2^8} \cup \{\varnothing\} \to \{-1, 0, 1\}^{10} \tag{4.25}$$

where we will write the output tuple from least to most significant bit, and $\varnothing$ is used to denote a "don't care" input symbol that should be not included in the output and is encoded as $e(\varnothing) = (0, 0, \ldots, 0)$. This is equal to the expected value of $e(b)$ averaged over a uniformly distributed $b \in \mathbb{Z}_{2^8}$.

Data-independence property

As already discussed, the balanced encodings do not depend on the running disparity. Therefore, the balanced encoding of any byte in the data stream only depends on that specific byte, and is unaffected by other surrounding values.

This means, for example, that the balanced encoding of a stream corresponding to a colour (r, g, b) can be thought of as separately applying the balanced encoding to the streams for r, g, and b, with the remaining two channels set to "nothing" in each, and taking the sum of the results.

Sub-block independence property

Since the running disparity only affects sub-blocks with unbalanced encodings, which are ignored in e, the encoding function can be written as a concatenation of two sub-encoders: a 5b/6b encoder that outputs the six low bits, and a 4b/6b encoder outputs the four high bits. Let us label these sub-encoders $e_5 : \mathbb{Z}_{2^5} \to \{-1, 0, 1\}^6$ and $e_3 : \mathbb{Z}_{2^3} \to \{-1, 0, 1\}^4$:

$$e(b) = e_3(\lfloor b/2^5 \rfloor) || e_5(b \bmod 2^5). \tag{4.26}$$

Complementation property

In both of the sub-encoder tables, if an input x is encoded as a balanced sub-symbol y, the complement of x will be encoded as the complement of y. Since we represent output bits as -1 and 1, this can be formally stated as

$$e_5(b \oplus \texttt{0x1f}) = -e_5(b) \tag{4.27}$$

$$e_3(b \oplus \texttt{0x07}) = -e_3(b) \tag{4.28}$$

The equations above hold even if the encoding of b is unbalanced, since $0 = -0$.

4.3.2 Template construction

A template for a balanced 8b/10b encoded bit stream $b[n]$ is an array of complex-valued samples describing the signal that the eavesdropper would receive from DisplayPort emissions of a cable transferring the bits $b[n]$. Let f_c be the centre frequency the eavesdropper's receiver is tuned to, f_s the sampling rate, and B the receiver bandwidth.

In the receiving band, we can model the received signal $s(b)[i]$ as a sinc interpolation of $b[n]$ (which is a sampled signal with rate f_b), assuming that the frequency response of the entire system (transmitter, propagation medium, and receiver) is flat over the frequency range $[f_c - B/2, f_c + B/2]$. The received signal $z(b)[i]$ can be obtained by

modelling the function of the software-defined radio receiver, as in Section 3.3.1. The receiver downconverts the signal $s(b)[i]$ by $-f_\mathrm{c}$, filters it with a lowpass antialiasing filter which has a cutoff frequency $B/2$, and samples the result at rate f_s.

Since both downconversion and resampling are linear, their composition, the template construction function $z(b)$, is also linear. In other words, if a bitstream $b[n]$ is represented as a sum of m terms $b_1[n], b_2[n], \ldots, b_m[n]$ such that $b[n] = b_1[n] + b_2[n] + \cdots + b_m[n]$, the corresponding template can be computed by separately constructing templates for the individual terms:

$$z(b)[n] = z(b_1)[n] + z(b_2)[n] + \cdots + z(b_k)[n] \tag{4.29}$$

While template construction is linear for any input, the algorithms in this chapter use only non-overlapping sub-bitstreams, such that for any n at most one of $b_k[n]$ is non-zero (i.e. each bit is assigned to exactly one b_k).

4.3.3 Sub-templates

Before scrambling, the bytes in a DisplayPort stream can be classified into five categories: three colour channels (red, green, and blue), zero-valued fill bytes in fill and blanking regions, and control symbols (*fill start*, *fill end*, etc.). A template containing expected emissions for a region filled by a single colour can be described by a colour triple (r, g, b) and the scrambler position i_s at the beginning of the template.

Since we only reset the scrambler position at *scrambler reset* symbols, which are always the first byte of a horizontal blanking region, the scrambler position provides enough information to predict the location of fill regions. Let $t[n]$ be the array of predicted byte types, where $t[n] \in \{\mathrm{r}, \mathrm{g}, \mathrm{b}, \mathrm{f}, \mathrm{c}\}$ is the type of the n-th byte in the template: red, green, blue, fill, and control, respectively. If we label the scrambling byte sequence $\Xi[n]$, the byte stream used for template construction, after scrambling and before 8b/10b encoding, is

$$b_{rgb}[n] = \begin{cases} r \oplus \Xi[i_\mathrm{s} + n], & \text{if } t[n] = \mathrm{r} \\ g \oplus \Xi[i_\mathrm{s} + n], & \text{if } t[n] = \mathrm{g} \\ b \oplus \Xi[i_\mathrm{s} + n], & \text{if } t[n] = \mathrm{b} \\ \Xi[i_\mathrm{s} + n], & \text{if } t[n] = \mathrm{f} \\ \varnothing, & \text{if } t[n] = \mathrm{c} \end{cases} \tag{4.30}$$

The byte stream does not include control symbols, since all control symbols have unbalanced encodings. And as described above, the templates I use only include balanced encoded symbols.

Using the data-independence property of our balanced encoding, we can split $b_{rgb}[n]$ into three colour terms $b_r[n]$, $b_g[n]$ and $b_b[n]$, and a fourth term containing fill bytes $b_\mathrm{f}[n]$:

$$e(b_{rgb}[n]) = e(b_r[n]) + e(b_g[n]) + e(b_b[n]) + e(b_\mathrm{f}[n]) \tag{4.31}$$

where the red term is

$$b_r[n] = \begin{cases} r \oplus \Xi[i_\mathrm{s} + n], & \text{if } t[n] = \mathrm{r} \\ \varnothing, & \text{otherwise} \end{cases} \tag{4.32}$$

and the others are defined analogously.

We can further split the templates for the three colour terms into a 5b/6b and a 3b/4b sub-template. As an example, let us consider the red channel. We can write the colour as $r = 2^5 r_\mathrm{h} + r_\mathrm{l}$, where $r_\mathrm{l} \in \mathbb{Z}_{2^5}$ are the five low bits of r that will be encoded with the 5b/6b sub-encoder, and $r_\mathrm{h} \in \mathbb{Z}_{2^3}$ are the three high bits that will be encoded with the 3b/4b sub-encoder. In the same way, we can split the scrambler $\Xi[n] = 2^5 \Xi_\mathrm{h}[n] + \Xi_\mathrm{l}[n]$ into high and low bits. Red channel bits in 5b/6b-encoded blocks only depend on r_l, and those in 3b/4b-encoded blocks only depend on r_h. We can separately compute the 5b/6b blocks b_{r_h} for all r_h

$$b_{r_\mathrm{h}}[n] = \begin{cases} r_\mathrm{h} \oplus \Xi_\mathrm{h}[i_\mathrm{s} + n] & t[n] = \mathrm{r} \\ \varnothing & \text{otherwise} \end{cases} \tag{4.33}$$

and analogously the 3b/4b blocks b_{r_l} for all r_l.

Using the sub-block independence property, we can now encode $e(b_r)$ by separately encoding these two:

$$e(b_r) = e_5(b_{r_\mathrm{l}}) + e_3(b_{r_\mathrm{h}}). \tag{4.34}$$

The number of sub-templates can be further optimised using the complementation property. It is sufficient to encode b_{r_h} for those r_h where the top bit is zero, i.e. $0 \leq r_\mathrm{h} < 2^2$, and similarly only $0 \leq r_\mathrm{l} < 2^4$. For $r_\mathrm{h} \geq 2^2$, let $\hat{r}_\mathrm{h} = r_\mathrm{h} \oplus 7$ be its complement. From Equation 4.33, we can see that $b_{r_\mathrm{h}} = b_{\hat{r}_\mathrm{h}} \oplus 7$ (or they are both $\varnothing$). By the complementation property, then,

$$e_3(b_{r_\mathrm{h}}) = -e_3(b_{\hat{r}_\mathrm{h}}) \tag{4.35}$$

Using everything described above, we can write the balanced encoding of DisplayPort data $e(b_{rgb}[n])$ as a sum of seven separately encoded sub-bitstreams: two each (3b/4b and 5b/6b) for the red, green, and blue channel, and one for fill bytes. Since template construction is linear, we can create separate sub-templates for each of these terms, and express $z(e(b_{rgb}[n]))$ as their linear combination. We compute the following sub-templates:

$$z_{a_\mathrm{h}} = z(e_3(b_{a_\mathrm{h}})) : a \in \{r, g, b\}, 0 \leq a_\mathrm{h} < 2^2 \tag{4.36}$$

$$z_{a_\mathrm{l}} = z(e_5(b_{a_\mathrm{l}})) : a \in \{r, g, b\}, 0 \leq a_\mathrm{l} < 2^4 \tag{4.37}$$

$$z_\mathrm{f} = z(e(b_\mathrm{f})) \tag{4.38}$$

With these precomputed, any $z(e(b_{rgb}))[n]$ can be computed by adding seven terms:

- Split each channel byte into a 3b and a 5b value, e.g. $r = 2^5 \cdot r_\mathrm{h} + r_\mathrm{l}$.

- If $r_\mathrm{h} \geq 2^2$, replace it with $r_\mathrm{h} \oplus 7$ and set $f_{\mathrm{r_h}} = -1$. Otherwise, set $f_{\mathrm{r_h}} = 1$.
- Similarly, if $r_\mathrm{l} \geq 2^4$, replace it with $r_\mathrm{l} \oplus 31$ and set $f_{\mathrm{r_l}} = -1$. Otherwise, set $f_{\mathrm{b_l}} = 1$.
- Apply the same to the green and blue channel to obtain $0 \leq r_\mathrm{l}, g_\mathrm{l}, b_\mathrm{l} < 2^4$, $0 \leq r_\mathrm{h}, g_\mathrm{h}, b_\mathrm{h} < 2^2$, and sign indicators $f_{\mathrm{r_h}}, f_{\mathrm{g_h}}, f_{\mathrm{b_h}}, f_{\mathrm{r_l}}, f_{\mathrm{g_l}}, f_{\mathrm{b_l}}$.
- Finally:

$$\begin{aligned} z(e(b_{rgb}))[n] = z_\mathrm{f}[n] + \\ & f_{\mathrm{r_l}} z_{r_\mathrm{l}}[n] + f_{\mathrm{r_h}} z_{r_\mathrm{h}}[n] + \\ & f_{\mathrm{g_l}} z_{g_\mathrm{l}}[n] + f_{\mathrm{g_h}} z_{g_\mathrm{h}}[n] + \\ & f_{\mathrm{b_l}} z_{b_\mathrm{l}}[n] + f_{\mathrm{b_h}} z_{b_\mathrm{h}}[n]. \end{aligned} \tag{4.39}$$

This requires $2^2 + 2^4 = 20$ sub-templates each for the red, green, and blue channel, and one sub-template for fill bytes, so a total of $3 \cdot 20 + 1 = 61$ sub-templates.

4.3.4 Cross-correlation

For colour identification or image reconstruction, the eavesdropper is interested in finding the RGB triple (r, g, b) that maximises the magnitude of the complex-valued dot product $x_{rgb} = \sum_{n \in \mathbb{W}} s[n] z_{rgb}[n]^*$ of the received signal $s[n]$ and template $z_{rgb}[n]$ over a window $\mathbb{W}$ containing samples of interest. Let us compute the dot product of $s[n]$ with each of the 61 sub-templates:

$$x_a = \sum_{n \in \mathbb{W}} s[n] z_a[n]^*. \tag{4.40}$$

Using linearity of dot products, we can write x_{rgb} as a sum of seven complex numbers:

$$x_{rgb} = x_\mathrm{f} + f_{\mathrm{r_h}} x_{r\mathrm{h}} + f_{\mathrm{r_l}} x_{r_\mathrm{l}} + \cdots + f_{\mathrm{b_l}} x_{b_\mathrm{l}}. \tag{4.41}$$

4.3.5 Enumeration

Given the 61 dot products x_a, we could directly find the colour with the highest correlation as the one maximising x_{rgb}, using Equation 4.41 to evaluate it efficiently:

$$(r, g, b) = \operatorname*{argmax}_{0 \leq r,g,b \leq 255} |x_{rgb}| \tag{4.42}$$

If the eavesdropper knows that the target image is grayscale, the most likely colour or a list of colours sorted by x_{rgb} can be found quickly by checking the 256 possible triples.

For colour images, testing all 2^{24} possible RGB values takes approximately 0.5 seconds on the laptop I used. This is acceptable for a one-time colour identification for a single

image region, but is not practical if there are many such regions of interest. For example, in my evaluation dataset, I attempt to identify the colour of each image line, and so have 628 such regions per frame, or 37680 per second.

To speed up the search, I first compute sub-template correlations only using fill bytes and one of the colour channels. For example, for the red channel, these would be

$$|x_{\mathrm{f}} + f_{\mathrm{r_h}} x_{r_{\mathrm{h}}} + f_{\mathrm{r_l}} x_{r_{\mathrm{l}}}| \tag{4.43}$$

I then only include the K candidates for r with highest correlation, and similarly select top K green and blue bytes. I then find the final (r, g, b) triple using the same correlation maximisation approach as above, limited to K^3 candidate values.

4.3.6 Averaging

Colour enumeration accuracy can be improved by assuming that the region of interest does not change between frames, similarly to how periodic averaging is used to reduce the noise level in electromagnetically eavesdropped images (Section 3.4). Since this tracking algorithm does not align the phase of samples in $s[n]$ with the template, an unknown phase shift exists between dot products x_a in differing frames. They, therefore, cannot be averaged coherently, and only the absolute value computed in Equation 4.42 can be replaced by an average over the frames.

Let $x_{rgb}[i]$ be the correlation for the region of interest in the i-th frame. I search for the colour maximising the average (equivalently, total) correlation magnitude over all frames:

$$(r, g, b) = \underset{0 \le r,g,b \le 255}{\operatorname{argmax}} \sum_i |x_{rgb}[i]| \,. \tag{4.44}$$

The top-K optimisation described above can similarly use averaged correlation magnitude instead of a single value.

4.3.7 Implementation

In my implementation of the colour enumeration algorithm, the most computationally expensive step is sub-template resampling from f_{b} to f_{s}. My initial implementation using the resampling function from the DSP.jl Julia package took 23 ms to resample the bit-stream for a single image line on an Intel i5-6300U processor. Only 1.5 ms of this was the resampling operation, and the rest of the time was spent computing coefficients for the antialiasing filter.

If we use the tracked DisplayPort bitrate, the resampling ratio $f_{\mathrm{b}}[i]/f_{\mathrm{s}}$ will change every line, and so the resampling filters could only be cached between sub-templates in a single line. The processing time is further increased because the ratio is real-valued, and not

a rational number with a small numerator and denominator, for which more efficient resampling algorithms can be used. Because of this, I chose to only use the tracked bitrate to maintain the scrambler position $i_\mathrm{s}[i]$, and use the nominal bitrate $f_\mathrm{b} = 1.62$ GHz for template construction.

For the 800×600 @ 60.3 fps video mode used for evaluation, a padded line is on average $n_\mathrm{p} = 4276.8$ bytes long. Assuming that the real bitrate differs from the nominal value by at most $\delta_\mathrm{f} = 30$ ppm, the difference between the predicted scrambler position at the last byte using nominal or tracked bitrate would be at most $\delta_\mathrm{f} n_\mathrm{p} = 0.13$ bytes. This is similar in magnitude to the accuracy obtained by PID-based scrambler code phase tracking, and (as can be seen in Figure 4.3) still in the region where the correlation of the band-limited signal and template is large.

I further simplify resampling by upsampling $s[i] = 50$ MHz from f_s to the first larger divisor of $f_\mathrm{b} = 1.62$ GHz, which is $f_\mathrm{b}/32 = 50.625$ MHz. This one-time operation simplifies the resampling ratio to $1 : 32$, a decimation operation which can be implemented more efficiently.

With the optimisations described so far, using the resampling function from DSP.jl, resampling a single template took 600 µs. I further improve this to 230 µs by first decimating the input by $1 : 16$ using a fast *cascaded integrator-comb* (CIC) filter, and using the DSP.jl FIR-based resampling in a second $1 : 2$ decimation stage. I do not use the CIC filter for the entire $1 : 32$ resampling, since it introduces significant frequency distortion in higher frequencies.

For the CIC filter with decimation rate $R = 16$, I use $N = 3$ stages and a $M = 1$ lag. After decimating, I apply a 3-coefficient FIR filter with taps $[-1/4, 3/2, -1/4]$, suggested by Jovanovic Dolecek and Mitra as a compensating filter that produces a more flat response up to 3/5 of the Nyquist rate (i.e. up to $6/5 f_\mathrm{s}$) [77].

4.4 Experimental evaluation

The goal of this evaluation is to test the performance of colour enumeration over a single image line, where the eavesdropper's primary goal is to find the background colour. As shown by demonstrations in Section 3.5.4, knowing this is enough to recover readable text, since the letters are visible as negatives in the background channel. The foreground channel is not needed for successful eavesdropping, and for anti-aliased text contains much less human-readable information.

The dataset includes both grayscale and colour images. I include both single-colour lines and lines containing text, which would be present in at least a part of a realistic image and represent a more challenging target (since not all pixels have the same colour).

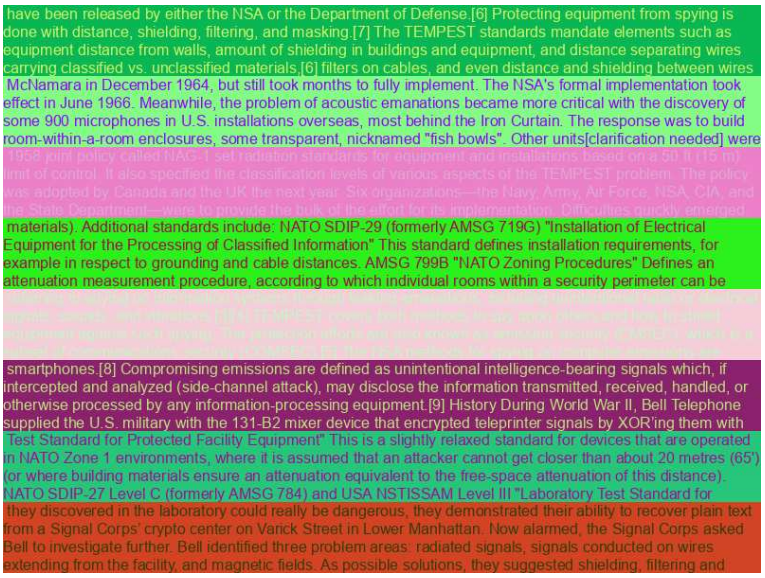


Figure 4.6: An example colour image with text used for evaluation. Text was taken from the "TEMPEST" Wikipedia page [78].

4.4.1 Images

The dataset contains 214 images, split into four categories:

- 32 grayscale images without text.
- 75 colour images without text.
- 32 grayscale images with text.
- 75 colour images with text.

Each 800×600 test image is split into eight horizontal 800×75 strips. Each strip is assigned a single background colour and, if the image contains text, a text colour. The strip is entirely filled with the background colour, with random text chosen to fill the entire strip drawn on top using 16 pixel-per-em Arial, with antialiasing enabled. The text was rendered using the Python library Pillow 10.0. A sample colour image with text is shown in Figure 4.6.

Each of the 256 grayscale colours is used once as a background colour in images without text, and once each as a background and a text colour in images with text. The colours were randomly assigned to strips, such that the background and text colours are different, but without any other restrictions such as a minimum contrast.

The background and text colours for colour images were chosen uniformly at random. Again, the only restriction was that they should not be exactly equal (at least one of the red, green, or blue values must differ).

Figure 4.7: Antenna and monitor used to record the evaluation dataset.

4.4.2 Setup

The eavesdropping target used for my evaluation was again a laptop with Intel Skylake GT2 graphics controller, connected via a miniDP-to-DP adapter and a 1.5 m DisplayPort cable to an iiyama ProLite XUB2495WSU LCD monitor (as in Section 3.5.1) configured with the VESA DMT 800×600 @ 60.3 fps video mode.

I recorded the emissions using a log-periodic antenna (0.2–3 GHz, Schwarzbeck VULSP 9111B) directly connected via a 4.6 m long RG-213/U coax cable to an Ettus USRP X300 software-defined radio receiver with Ettus UBX-160 daughterboard. The receiver was tuned to a centre frequency of $f_c = 400$ MHz, with sampling rate $f_s = 50$ MHz, streaming float32 IQ values to a desktop computer via a 10GBASE-SR optical-fibre link.

All devices (software-defined radio, laptop, monitor) had been powered up at least 30 minutes before I started recording, to allow their temperature to stabilise and reduce any temperature-related changes in oscillator frequency.

I made all recordings in the corridor of a university building, without any measures to shield the setup from other signal sources. A photo of the setup (antenna and target) can be seen in Figure 4.7.

4.4.3 Recordings

For each image and antenna distance, I made a 3 s recording. I extracted scrambler reset times and computed timing parameters for the reset model from the entire recording. To reduce the size of individual recordings and allow me to include more images in the dataset, after computing the timing parameters, I saved only 0.2 s of recorded emissions (12 frames), starting from 1 s into the recording.

To detect any outliers, e.g. due to a bursty radio transmission in the band I was recording in, I made two validation checks for each recording. The first was to test if reset model synchronisation was successful, which I accepted if the mean square error between observed and predicted reset times was under 2 samples. The second check was a comparison between the Welch periodogram for the current recording and the ten preceding recordings. I computed a 2048-point periodogram with 1024-sample overlaps, and labelled the recording as an outlier if any frequency bin in the power spectrum was 7 dB above the mean value in the previous ten recordings.

In the 642 recordings, there was only a single outlier, for which synchronisation was unsuccessful. I re-recorded it, and used the new (successful) one for evaluation.

4.4.4 Results

I evaluated the colour enumeration algorithm by measuring the success rate when identifying the background colour from a single image line, using images in the dataset described above. I computed a sorted list of the 1000 highest-correlation colours (256 for grayscale images) separately for each line. Since each recording is 100 ms long (12 frames), the output for a recording consists of $12 \times h_\mathrm{t} - 1 = 7535$ such lists (minus one since the recording does start on a line boundary). I also computed one list of candidate colours for each line from all 12 occurrences of that line in the recording, using the averaging approach described in Section 4.3.6.

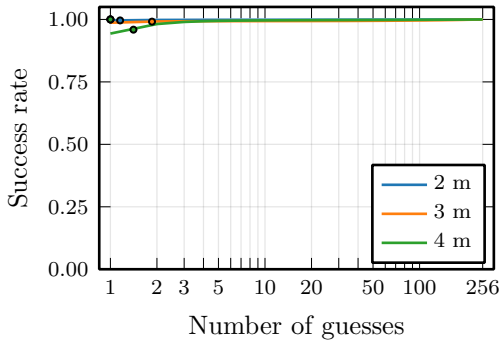
The timing parameters and tracking computed by my algorithm only provide horizontal alignment, and are insufficient to match the identified line colours with correct values: without vertical alignment, this pairing is only known up to a cyclic shift. I align the algorithm output with the correct colours by selecting the shift that maximises the number of lines where the highest-correlation colour is correct.

Table 4.2 shows the fraction of lines where the highest-correlation colour returned by the colour enumeration algorithm is the displayed background colour.

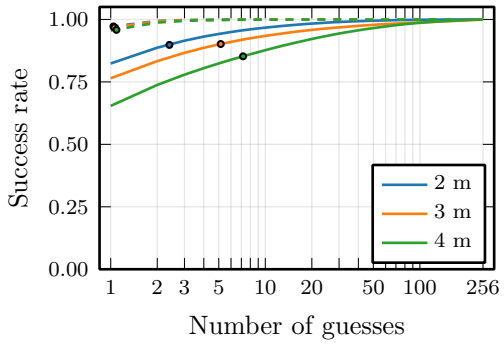
In practice, an eavesdropper does not need to limit demodulation to the first colour returned by the algorithm. One could select several top colours, reconstruct the image channel for each of them, and then have the human operator (or an image quality metric) select the colour that results in a meaningful image. In Figure 4.8, I show the average

Table 4.2: Percentage of lines where the most likely colour returned by the colour enumeration algorithm was correct, averaged over same-category images in the dataset for each antenna distance.

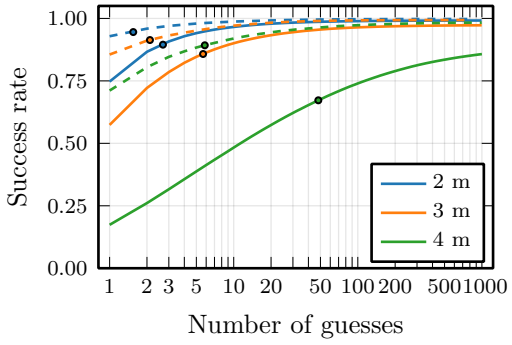
Image category	Single line			12-frame average		
	2 m	3 m	4 m	2 m	3 m	4 m
Grayscale, no text	99.6%	98.7%	94.3%	100%	100%	100%
Grayscale, with text	82.4%	76.5%	65.4%	97.1%	96.7%	95.6%
Colour, no text	74.7%	57.3%	17.4%	92.9%	85.6%	71.1%
Colour, with text	39.2%	30.0%	8.9%	51.9%	48.4%	38.3%



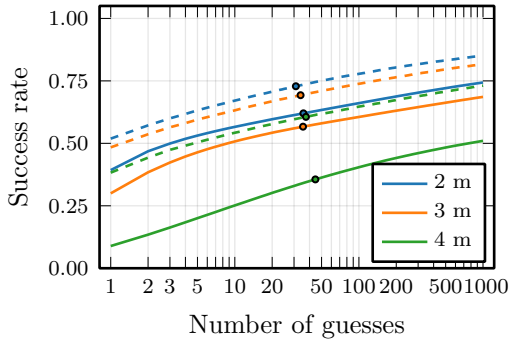
(a) Grayscale, no text



(b) Grayscale, with text



(c) Colour, no text



(d) Colour, with text

Figure 4.8: Average rank of correct line colour for single-line enumeration (solid lines) and 12-frame averaged correlation (dashed lines). Markers denote the guessing entropy (average rank of correct value).

rank of the correct colour, i.e. the average number of channels the eavesdropper would need to examine before encountering the correct one. This metric is commonly used in side-channel research, where the average rank of the correct value is known as the *guessing entropy* [79, 80].

Chapter 5

Image reconstruction using phase tracking

In Chapter 3, I described a DisplayPort image reconstruction algorithm for which I made two simplifying assumptions: that the lane bitrate is constant and equal to the nominal value, and that every 512-line interval between scrambler resets has the same length. With these assumptions, I described times at which the scrambler is in the initial state with a three-parameter reset model, which I used to align a synthesised scrambling sequence with the received signal. I showed that it is possible to classify pixel colours in the eavesdropped image by comparing the magnitude of short-term correlations of the received signal and templates synthesised using the aligned scrambler. The algorithm required lengthy averaging times to reduce noise, as well as a correlation window that resulted in a horizontally blurred image in which narrow features (e.g. vertical lines in letters) were difficult to distinguish. This was in part necessary because the simplified scrambler reset model is not sufficient to fully accurately align templates, and so the computed correlations were lower off-peak values.

In Chapter 4, I showed that the received signal and scrambler can be accurately aligned using a delay-locked loop. This improves the signal-to-noise ratio in signal-template correlations, and also provides information about the current pixel's horizontal position in the received signal, which can be used to include fill bytes in the templates. With these improvements, I could efficiently identify candidate colours in the image. I will now return to the image reconstruction problem, and describe a more sophisticated eavesdropping algorithm that uses scrambler code phase and carrier phase tracking to enable coherent averaging of template correlations. The improved signal-to-noise ratio addresses both weaknesses of the algorithm in Chapter 3: a clear image can be recovered with significantly lower averaging times and without a correlation window, i.e. without horizontal blurring.

Following from the previous chapters, I will start with a delay-locked loop that tracks the scrambler position, and extend it with a phase correction based on the tracked position

that approximately aligns signal and template phases. Next, I measure the phase alignment error after each image line, and use a *phase-locked loop* to compute a more precise phase correction (Section 5.1). I then construct phase-aligned templates for each colour in the image, and compute a correlation in a similar way to the previous eavesdropping algorithm. Instead of averaging correlation magnitudes as before, phase alignment allows *coherent averaging* (Section 5.2) of complex-valued correlations. This reduces the noise level in the output, and does not discard sign information, which lets the eavesdropper distinguish between positive and negative correlations, improving contrast for complementary colour pairs.

After computing the averaged template products, I apply a final phase adjustment to remove any constant offset or drift within a line, between updates to the phase-locked loop (Section 5.3). I also suggest an interpolation method that rescales the image data in the recovered output to the original resolution while suppressing fill regions (Section 5.4).

Section 5.5 presents a practical demonstration of the eavesdropping attack in which I reconstruct readable text from 8 m distance from the target monitor. I discuss the effect of increasing averaging time and show that it is possible to compute a coherent average of multiple recordings made even multiple minutes apart. I also discuss factors limiting the maximum eavesdropping distance, and provide an estimate for the longest distance at which my implementation of code and carrier phase tracking would work for the antenna and receiver I used. I also discuss how image characteristics such as choice of font, textured backgrounds, and partially static images influence the resulting image, and demonstrate successful 256-colour image reconstruction at 2 m distance.

Finally, in Section 5.6, I briefly describe two practical enhancements based on scrambler tracking: automated vertical alignment if the eavesdropping target is one of the Intel video controllers I tested, and an optimised implementation for black-and-white images.

5.1 Scrambler carrier phase tracking

Let us start with a brief recap of the scrambler tracking and colour enumeration algorithm from the previous chapter. Given a received IQ signal $s[n]$, centred at frequency f_c, with sampling rate f_s (a divisor of the DisplayPort bitrate f_b), I used a delay-locked loop to compute an array of scrambler code phase estimates $i_\mathrm{s}[n]$, meaning that at the time when sample $s[n]$ was taken, the scrambler produced the $i_\mathrm{s}[n]$-th byte. This index is real-valued, since samples need not be aligned with data bytes.

I then used this information to create templates for pixel colours or fill bytes by scrambling and encoding data, and then downconverting and resampling to match the behaviour of a software-defined radio receiver. The templates were aligned with $s[n]$ in time using the scrambler indices $i_\mathrm{s}[n]$, but not in phase of individual samples: all templates were

generated with the same starting phase (specifically, the first sample was real-valued). The phase of the complex-valued signal-template correlation was therefore not meaningful, and I could only average the magnitude after discarding phase information.

In GPS literature, the phase correction applied to the local code replica is referred to as *carrier phase*. Although electromagnetic emissions of a DisplayPort interface do not contain an explicit carrier, the tracking problem described in this section is analogous to GPS carrier phase tracking, and I will analogously refer to the phase adjustment applied to template samples using the same term.

In this section, I will describe a two-step algorithm that adjusts carrier phase to match the received signal by first applying an approximate correction based on $i_\mathrm{s}[n]$, followed by a fine-tuning phase-locked loop. The resulting template should be phase-aligned with the signal, meaning that for a part of the received signal $s[n], s[n+1], \ldots, s[n+m]$ (e.g. one image line) and the corresponding template $z[i]$, their dot product should be real-valued:

$$\angle \left(\sum_{i=0}^{m} s[n+i] z[i]^* \right) \approx 0. \tag{5.1}$$

In practice, an eavesdropper cannot measure the carrier phase error directly, since they do not (yet) know the image contents and therefore cannot synthesise $z[i]$. I will therefore only include fill bytes in the template when estimating the carrier phase error using a discriminator $e_\varphi[n]$, which I will use as feedback for the phase-locked loop

$$e_\varphi[n] = \angle \left(\sum_{i=0}^{m} s[n+i] z^\mathrm{f}[i]^* \right) \tag{5.2}$$

and minimise $|e_\varphi[n]|$.

5.1.1 Approximate alignment

Consider a template for an image region starting at sample n, at which time the scrambler index is $i_\mathrm{s}[n]$. Since the template is created by downsampling a downconverted bit sequence, in which each sample is one DisplayPort bit, the scrambler position at the first template sample is rounded to the nearest bit $\lfloor 10 i_\mathrm{s}[n] + 0.5 \rfloor$. Assuming for a moment that the DisplayPort bitrate f_b is constant and exactly equal to the nominal value, we can compute the time at which the template begins as $\lfloor 10 i_\mathrm{s}[n] + 0.5 \rfloor / f_\mathrm{b}$, and the corresponding phase shift for a signal centred at f_c:

$$2\pi f_\mathrm{c} \lfloor 10 i_\mathrm{s}[n] + 0.5 \rfloor / f_\mathrm{b}. \tag{5.3}$$

Using the nominal bitrate is usually not accurate enough for this phase correction to be useful. For example, for a monitor using the 800×600 @ 60.3 fps video mode, one image

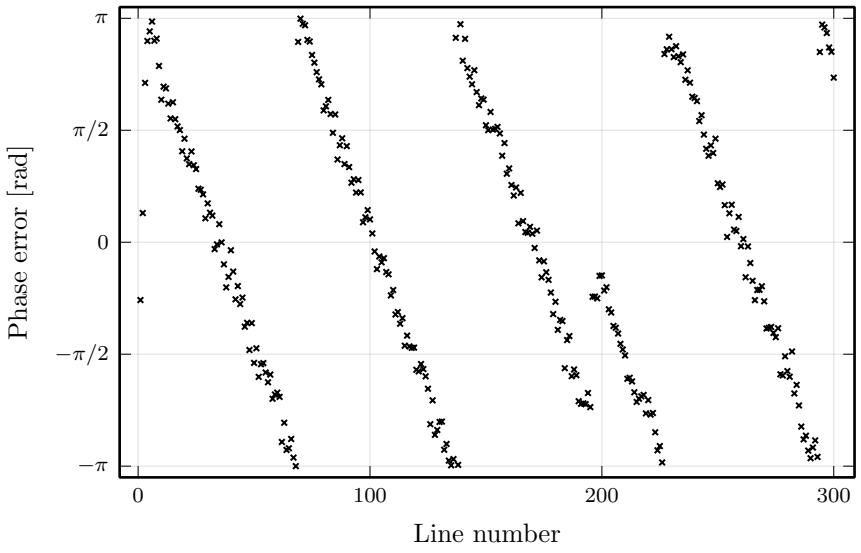


Figure 5.1: Plot of carrier phase error $e_\varphi[n]$ for a recording made at 8 m antenna distance, after initial correction but without tracking.

line takes around 26.5 µs to transfer, corresponding to a $\Delta\varphi = 2\pi \cdot 400\ \text{MHz} \cdot 26.5\ \mu\text{s} \approx 66601$ rad phase change. A 10 ppm change in f_b (a somewhat optimistic estimate) would introduce a $10 \cdot 10^{-6} \cdot \Delta\varphi \approx 0.67$ rad phase jump between templates for adjacent lines.

I instead use an average $\bar{f}_\text{b}$ of the estimate $f_\text{b}[n]$ computed by the scrambler code tracking delay-locked loop. I use the same averaged rate to compute the phase correction for all lines to keep the rate at which the error accumulates stable and independent of changes in $f_\text{b}[n]$, which simplifies later PID-based tracking:

$$\bar{f}_\text{b} = \frac{1}{|s|} \sum_{n=1}^{|s|} f_\text{b}[n] \tag{5.4}$$

I then compute the approximate carrier phase correction $\Delta_\varphi[n]$ from Equation 5.3, using the averaged bitrate:

$$\Delta_\varphi[n] = 2\pi f_\text{c} \left\lfloor 10 i_\text{s}[n] + 0.5 \right\rfloor / \bar{f}_\text{b} \tag{5.5}$$

The correction can then be applied directly to templates by multiplying each sample by $\text{e}^{\text{j}\Delta_\varphi[n]}$, or more efficiently by just multiplying any correlations with that template by the conjugate $\text{e}^{-\text{j}\Delta_\varphi[n]}$.

Figure 5.1 shows an example of the remaining carrier phase error $e_\varphi[n]$ after applying this correction. A small amount of drift due to the difference between $\bar{f}_\text{b}$ and the true value remains, as well as an unknown initial phase offset between $s[n]$ and the templates. The jump around line 200 is due to a mispredicted scrambler reset caused by a leap byte.

5.1.2 Phase-locked loop

After approximate phase correction, I use a phase-locked loop to eliminate the remaining drift and the initial offset, so that template correlation is real-valued (i.e. $e_\varphi[n] = 0$). The structure of this controller is similar to that used for scrambler code phase tracking (Section 4.2.3): I measure the carrier phase error, lowpass filter it, and use the result as an input to a PID controller that computes a fine-tuning adjustment $\Delta'_\varphi[n]$.

Every line, I synthesise a fill byte template $z^\mathrm{f}[i]$ (code-aligned using $i_\mathrm{s}[n]$), and shift its phase in two steps: by the coarse adjustment $\Delta_\varphi[n]$ and the previous output of the phase-locked loop $\Delta'_\varphi[n-1]$. I then estimate the carrier phase tracking error as

$$e_\varphi[n] = \angle\left(\mathrm{e}^{\mathrm{j}(-\Delta_\varphi[n]-\Delta'_\varphi[n-1])}\sum_{i=0}^{m} s[n+i]z^\mathrm{f}[i]^*\right) \tag{5.6}$$

and lowpass filter it by computing an exponential average $\bar{e}_\varphi[n] = \tilde{\alpha}\bar{e}_\varphi[n-1]+(1-\tilde{\alpha})e_\varphi[n]$.

Note that $-\pi < e_\varphi[n] \leq \pi$, and an error near $\pm\pi$ can wrap around between lines. This is unlikely after the tracking loop locks onto the phase, but might happen in the beginning, and a robust averaging implementation should handle this case: for example, if $\bar{e}_\varphi[n-1] \approx \pi$ and $e_\varphi[n] \approx -\pi$, the unwrapped error $e_\varphi + 2\pi$ should be used instead.

I then compute the fine-tuning adjustment for the next line using a PID controller, discretised in the same way as the scrambler code phase tracking controller (Equation 4.11):

$$\Delta'_\varphi[k] = \tilde{K}_\mathrm{P}\bar{e}_\varphi[k] + \tilde{K}_\mathrm{I}f_\mathrm{h}^{-1}\sum_{m\leq k}\bar{e}_\varphi[m] + \tilde{K}_\mathrm{D}f_\mathrm{h}(\bar{e}_\varphi[k]-\bar{e}_\varphi[k-1]) \tag{5.7}$$

Figure 5.2 shows the resulting carrier phase error after applying the PID-based fine-tuning correction, which eliminates the drift and offset from Figure 5.1.

Coefficients

I chose the PID controller parameters using the same approach I used for the code phase tracking controller, a "black-box optimiser" provided by the BlackBoxOptim.jl Julia package, which uses an adaptive differential evolution optimiser. I used a recording from the colour enumeration dataset, made at 2 m antenna distance, and computed the signal-template carrier phase offset after initial correction $\angle\mathrm{e}^{-\mathrm{j}\Delta_\varphi[n]}\left(\sum_{i=0}^{m} s[n+i]z^\mathrm{f}[i]^*\right)$. I could then quickly evaluate the performance of a chosen set of PID parameters, without having to build any templates or compute correlations, since I could emulate the phase-locked loop behaviour by directly applying the correction to this error.

I used the optimiser to find sampling rate dependent parameters ($\tilde{K}_\mathrm{P}, \tilde{K}_\mathrm{I}f_\mathrm{h}^{-1}, \tilde{K}_\mathrm{D}f_\mathrm{h}$) and the exponential averaging coefficient $\tilde{\alpha}$ that minimise the average of the absolute carrier

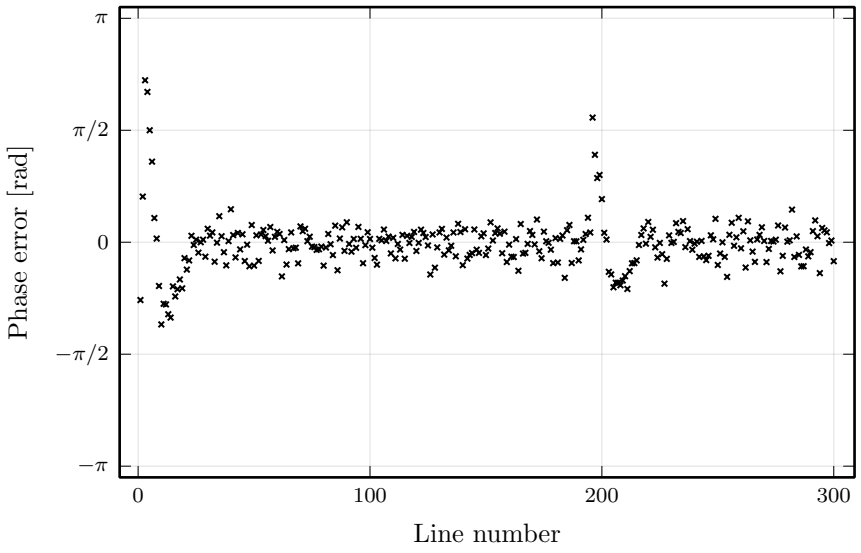


Figure 5.2: Plot of carrier phase error $e_\varphi[n]$ for a recording made at 8 m antenna distance, after PID-based tracking.

phase error $|e_\varphi[n]|$ over all image lines in the recording. The search range for the former was $[0, 50]$, and the range for $\tilde{\alpha}$ was $[0, 1]$.

The best-performing coefficients found were

$$(\tilde{K}_\mathrm{P}, \tilde{K}_\mathrm{I} f_\mathrm{h}^{-1}, \tilde{K}_\mathrm{D} f_\mathrm{h}, \tilde{\alpha}) = (24.09, 4.34, 6.88, 0.99). \tag{5.8}$$

The corresponding rate-independent coefficients, substituting $f_\mathrm{h} = 37.88$ kHz, are

$$(\tilde{K}_\mathrm{P}, \tilde{K}_\mathrm{I}, \tilde{K}_\mathrm{D}, \tilde{\alpha}) = (24.09, 164.4 \text{ kHz}, 0.18 \text{ kHz}^{-1}, 0.99). \tag{5.9}$$

We can compute the loop bandwidth following the same approach as Section 4.2.4. The transfer function of the lowpass filter is

$$G_1(z) = \frac{1 - \tilde{\alpha}}{1 - \tilde{\alpha} z^{-1}}, \tag{5.10}$$

and the PID controller

$$G_2(z) = \tilde{K}_\mathrm{P} + \tilde{K}_\mathrm{I} f_\mathrm{h}^{-1} \frac{1}{1 - z^{-1}} + \tilde{K}_\mathrm{D} f_\mathrm{h} (1 - z^{-1}). \tag{5.11}$$

Since this PLL directly outputs a phase correction, rather than a frequency correction, there is no final integrator and the open-loop response is $G(z) = G_1(z) G_2(z)$.

The 3 dB loop bandwidth $f_{3\,\mathrm{dB}}$, at which the closed-loop frequency response is $1/\sqrt{2}$, is $f_{3\,dB} \approx 2.06$ kHz, or 5% of the line rate. With my carrier tracking implementation, much narrower bandwidths are undesirable since they would result in a long impulse response, and therefore a long settling time after a scrambler reset (Figure 5.1).

5.2 Coherent averaging

Now that we can align the phase of template samples with the received signal, we can revisit the image reconstruction algorithm described in Section 3.4. Recall that I had assumed that the target image mostly consisted of a small set of colours $\mathbf{C} = \{c_1, c_2, \ldots, c_n\}$, constructed a pair of templates z_k^+, z_k^- for each colour c_k, and computed a score $u_k[n]$ for each received sample $s[n]$ and colour c_k as a short-term correlation between the received signal and the two templates (Equation 3.14):

$$u_k[i] = \max_{z \in z_k^+, z_k^-} \left\{ \left| \sum_{l=0}^{w-1} s[i+l] z[i_\mathrm{s} + l]^* \right|^2 \right\} \tag{5.12}$$

Note that the z_k^+ and z_k^- are one scrambler period long and are entirely filled with colour bytes, unlike templates I used for tracking and colour enumeration, which are one line long and include fill regions.

The alignment-assisted image reconstruction algorithm similarly computes a score $u_k[n]$ for each input sample n and colour c_k, with three differences:

- Since $s[n]$ and z_k are now phase-aligned, periodic averaging can be done *coherently*, on the complex values instead of magnitudes, and the score $u_k[i]$ is therefore complex-valued.
- Accurate alignment and coherent averaging reduce the noise level enough that the short-term correlation window w is not necessary. A one-sample product ($w = 1$) resulted in clear images, without horizontal blurring that would occur with $w > 1$.
- Instead of comparing two templates corresponding to the two possible encoder states (positive and negative running disparity), I compute the product with a single balanced template z_k, such that the result can be averaged without terms due to opposite-disparity unbalanced symbols cancelling.

The score for colour c_k is then simply

$$u_k[n] = s[n] \tilde{z}_k[i_\mathrm{s}[n]]^* \tag{5.13}$$

where $\tilde{z}_k[n]$ is a template for a colour c_k that is time- and phase-aligned with $s[n]$, at scrambler position $i_\mathrm{s}[n]$. Here, I implicitly assume that the template was synthesised such that lines begin at positions predicted by code tracking; in practice, my implementation achieves this by computing $\tilde{z}_k[n]$ one line at a time when the line number corresponding to $i_\mathrm{s}[n]$ increments.

Finally, I average the scores u_k at the frame period, using $i_\mathrm{s}[n]$ to compute the image position (line number and horizontal offset) corresponding to each input sample n. The output is then a single complex-valued score $\bar{u}_k[i]$ for each colour c_k and image position i.

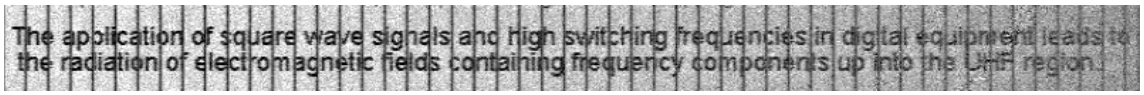


Figure 5.3: Example from a two-channel output image showing the difference between the white (c_1) and black (c_2) channel. Pixel brightness is proportional to $\Re\{\bar{u}_1[i] - \bar{u}_2[i]\}$, and the image is rescaled to the original aspect ratio after removing fill regions. The image was produced by averaging scores for a 2 s long signal (≈ 120 frames), recorded at 2 m antenna distance.

Since the carrier phase tracking controller adjusts template phase such that the expected correlation is real-valued and positive, I can now create an output image by assigning the colour with the largest real part $\operatorname{argmax}_k \Re\{\bar{u}_k[i]\}$ to each pixel (*voting mode*, as in Section 3.4). If the image mostly consists of only two colours c_1 and c_2, the output can alternatively be a grayscale image with pixel brightness proportional to $\bar{u}_1[i] - \bar{u}_2[i]$ (*silhouette mode*).

An example of a grayscale image produced by this algorithm, with fill bytes removed and rescaled to the original dimensions, is shown in Image 5.3.

5.3 Phase adjustment

The contrast between the two channels in Figure 5.3 is high at the beginning of each line, but fades towards the right side of the image due to phase drift within each line. Line templates are aligned with the received signal such that the total correlation over the fill regions is near-zero (Equation 5.2), but the phase difference between $s[n]$ and the template drifts slightly between samples in the line since the real DisplayPort bitrate differs from the nominal rate f_b used in template construction for efficiency.

Consequently, while signal-template correlation is on average real-valued, individual values $\bar{u}_k[n]$ will be offset in phase, and for them taking the real part results in a lower signal-to-noise ratio (up to no signal at all, if the phase drifts by $\pi/2$). We can measure the drift by plotting the angle $\angle\bar{u}_\mathrm{f}[i]$ at horizontal positions i inside fill regions, where $\bar{u}_\mathrm{f}[i]$ is the score for a template containing fill bytes averaged over all image lines (Figure 5.4).

Since the remaining phase error is largely due to a mismatch between the nominal and real bitrate, it is a linear function of the horizontal position I. I compensate for this drift by approximating the phase error as

$$\angle\bar{u}_\mathrm{f}[i] = \Delta\varphi_0 + k_\varphi i \tag{5.14}$$

and compute $\Delta\varphi_0$ and k_φ using a least-squares fit.

I then unrotate the scores for all channels by multiplying $\bar{u}_k[n]$ by $\mathrm{e}^{\mathrm{j}(-\Delta\varphi_0 - k_\varphi n_t)}$, where n_t

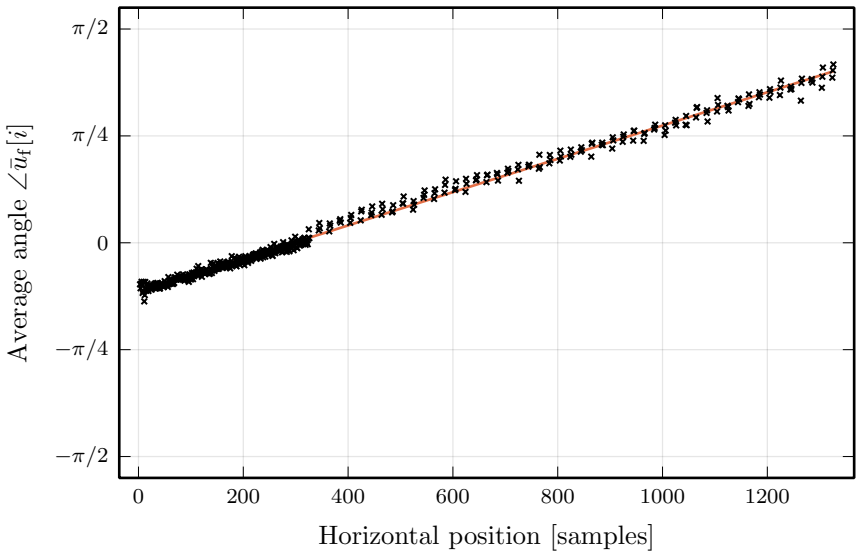


Figure 5.4: Plot of angle $\angle\bar{u}_\mathrm{f}[i]$ for the reconstructed image in Figure 5.3, averaged over all image lines, for horizontal positions i in the fill region.

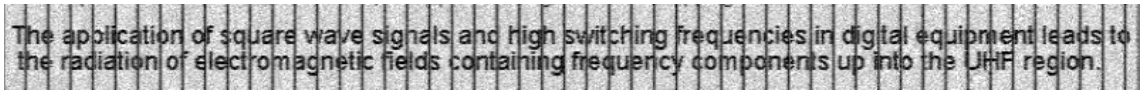


Figure 5.5: Output image sample for recording in Figure 5.3, after final phase correction.

is the horizontal image position for sample n, before creating an output image as before (Figure 5.5).

5.4 Removing fill regions

Rasterizing $\bar{u}_k[n]$ directly results in an image with height h_t and width $f_\mathrm{s}/(h_\mathrm{t}f_\mathrm{v})$ containing alternating image pixels and fill regions. Before producing the final output image, it should be rescaled to the displayed width w_d, and fill and blanking regions should be suppressed.

Let $\bar{u}_k[n]$ be a single line of colour scores with length L, and $\bar{u}'_k[n]$ the rescaled output with length w_d. If there were no fill regions, $\bar{u}'_k[n]$ could be computed by linearly interpolating $\bar{u}_k[n]$. With fill regions, my rescaling algorithm is still based on linear interpolation, with different weights assigned to computed scores $\bar{u}_k[n]$ based on the fraction of bytes covered by that value that are image bytes.

Let w_p be the length of a transferred line, including the blanking period, after padding. I compute these weights by starting with a length w_p array $p[n]$ containing a mask that includes image bytes: $p[n] = 1$ if the n-th byte in a line is part of an image pixel, and $p[n] = 0$ if it is part of a fill or blanking region. I then linearly rescale $p[n]$ to $w[n]$ to obtain the interpolation weights for $\bar{u}_k[n]$. Finally, I linearly interpolate $\bar{u}_k[n]$ to the length w_d

Algorithm 2 Output line rescaling and fill region suppression

procedure RESCALE($\bar{u}_k[n], w_\mathrm{d}, w_\mathrm{p}$)
  $p \leftarrow \{1$ if byte n in a line is an image byte, otherwise $0 \mid 1 \le n \le w_\mathrm{p}\}$
  $w \leftarrow \{0, 0, \ldots, 0\}$ ▷ Length-$|\bar{u}_k[n]|$ array of weights
  $\Delta m \leftarrow |w| / w_\mathrm{p}$
  for $1 \le n \le w_\mathrm{p}$ **do** ▷ Rescale mask $p[n]$ to weights $w[m]$
    $(m_1, m_2) \leftarrow (\lceil \Delta m \cdot n \rceil, \lceil \Delta m \cdot (n+1) \rceil)$ ▷ Indices in w covered by $p[n]$
    $k \leftarrow (m_1 - \Delta m \cdot n)/\Delta m$ ▷ Fraction of $p[n]$ covering $w[m_1]$
    $w[m_1] \leftarrow w[m_1] + p[n] \cdot k$
    $w[m_2] \leftarrow w[m_2] + p[n] \cdot (1 - k)$
  end for
  $\bar{u}'_k \leftarrow \{0, 0, \ldots, 0\}$
  $(n, r) \leftarrow (1, 3)$ ▷ Output index n and remaining weight r
  for $1 \le m \le |w|$ **do** ▷ Linearly interpolate $\bar{u}_k[n]$ to $\bar{u}'_k[n]$ using weights $w[n]$
    while $w[m] > 0$ **do**
      if $r = 0$ **then** ▷ Next image pixel
        $(n, r) \leftarrow (n + 1, 3)$
      end if
      $w_\mathrm{c} \leftarrow \min(w[m], r)$ ▷ Weight for $\bar{u}'_k[n]$
      $\bar{u}'_k[n] \leftarrow \bar{u}'_k[n] + w_\mathrm{c} \cdot \bar{u}_k[m]$
      $w[m] \leftarrow w[m] - w_\mathrm{c}$
    end while
  end for
  return $\bar{u}'_k$
end procedure

output $\bar{u}'_k[n]$, such that the total weight contributing to each output value is equal to 3, the number of image bytes in one pixel (Algorithm 2).

5.5 Practical demonstration

5.5.1 Setup

The eavesdropping target was the same laptop used in previous chapters, with Intel Skylake GT2 graphics controller, connected via a miniDP-to-DP adapter and a 1.5 m DisplayPort cable to an iiyama ProLite XUB2495WSU LCD monitor configured with the 800×600 @ 60.3 fps video mode. I left the cable hanging behind the monitor and did not adjust its position to potentially improve signal quality.

I recorded the monitor emissions using a Yagi antenna (340–360 MHz, Sinclair SY307-

Figure 5.6: Antenna and monitor used for practical demonstrations described in this chapter.

SF6SNM(ABK)), connected via a 4.6 m long RG-213/U coax cable and a 3 dB attenuator to an Ettus USRP X300 software-defined radio receiver with Ettus UBX-160 daughterboard. The receiver was tuned to a centre frequency of $f_c = 350$ MHz, with sampling rate $f_s = 50$ MHz, streaming float32 IQ values to a desktop computer via a 10GBASE-SR optical-fibre link.

All devices (software-defined radio, laptop, monitor) were powered up at least 30 minutes before I started recording, to allow their temperature to stabilise and reduce temperature-related changes in oscillator frequency.

I made all recordings in the corridor of a university building, without any measures to shield the setup from other signal sources. A photo of the setup can be seen in Figure 5.6.

I also tested the eavesdropping attack separately with one different DisplayPort cable, video interface (Intel Xeon E3-1200 graphics controller in a desktop computer), and monitor (LG 32UD59). The attack worked without modification in all cases, and I did not notice any significant differences when compared to the evaluation target.

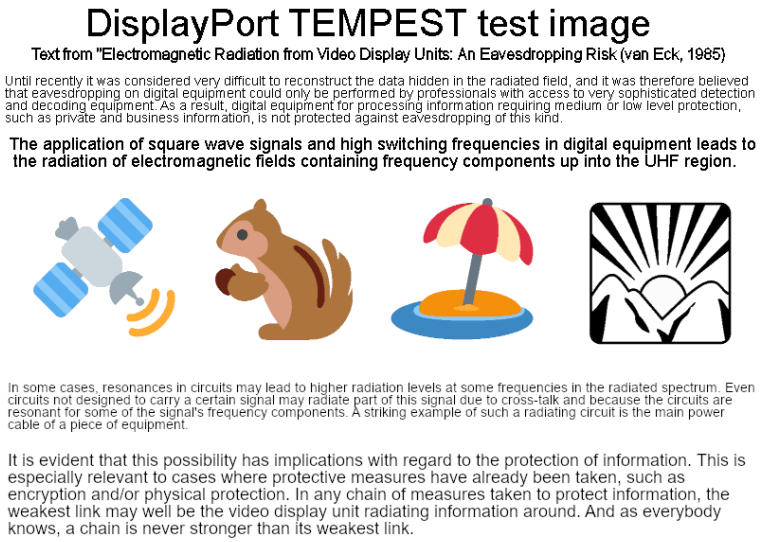


Figure 5.7: Test image containing four text paragraphs (Arial, top to bottom: without antialiasing, 14 and 18 pixels per em; with antialiasing, 14 and 18 pixels per em) and a 16-colour image. Text from van Eck [2]. Image from the Noto Emoji font.

5.5.2 Image

The test image (Figure 5.7) I used to showcase the eavesdropping attack includes multiple font sizes and rendering options, as well as the 16-colour image used to demonstrate the image reconstruction algorithm in Section 3.5.5. Two text paragraphs above the colour image were rendered with antialiasing disabled, and therefore only consist of two pixel colours: white (`#ffffff`) background and black (`#000000`) text. The remaining two paragraphs below the image use the same font size, but are antialiased.

5.5.3 Reconstructed images

Figure 5.8 shows the test image reconstructed from a 2 s long signal recorded at 8 m distance from the antenna to the target monitor, chosen as a distance near the upper bound at which initial scrambler timing parameters can be reliably estimated. Image reconstruction is done in voting mode, with pixels assigned the colour with the highest periodically averaged score $\bar{u}_k[n]$, out of the foreground, background, and sixteen colours in the emoji part of the image. Large single-colour areas are clearly visible, and while colour classification is noisy, the original colour of the areas can still be identified. Text with larger font size and without antialiasing is clear, and other text is harder to read from the colour output.

As was the case for the initial eavesdropping algorithm from Chapter 3, text can be made more readable by using silhouette mode and displaying the difference between the fore-

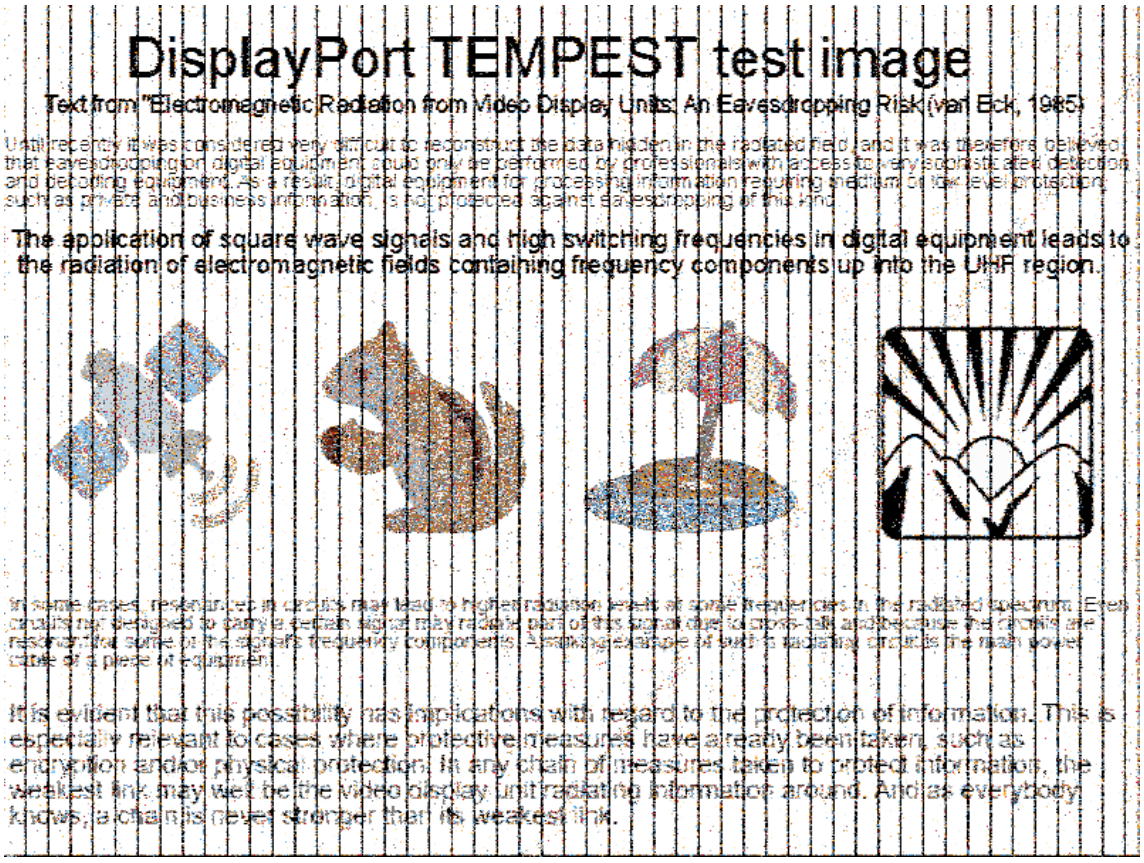


Figure 5.8: Test image reconstructed at 8 m antenna distance from a 2 s-long recording, with the highest-score colour $\operatorname{argmax}_k \bar{u}_k[i]$ assigned to each pixel (voting mode). Manually aligned vertically and trimmed to remove the horizontal blanking region.

ground and background channel $\bar{u}_1[n] - \bar{u}_2[n]$, instead of a hard-cutoff colour classification that includes colours that are only present in the illustration but not in text. Figure 5.9 shows this grayscale output for the same recording.

Silhouette mode is particularly helpful for antialiased text, where edge pixels are neither the background nor the text colour. They are therefore classified as the colour that happens to have the highest correlation with the pixel value, which is not necessarily visually similar. They are, however, clearly visible in the background channel as a negative image (Figure 5.10a).

The initial image reconstruction algorithm described in Chapter 3 produced horizontally blurred images due to a correlation window. This algorithm does not use one, and allows us to see vertical lines that are as thin as a single pixel (Figure 5.10b). The 50 MHz sampling rate used in this demonstration is near the $\frac{1.62\ \text{Gbit/s}}{30} = 54$ MHz padded pixel rate (shared by all video modes using the 1.62 Gbit/s data rate), and so an output pixel before rescaling is only slightly larger than a single displayed pixel.

As previously discussed in Section 3.5.3, complementary colours such as black and white

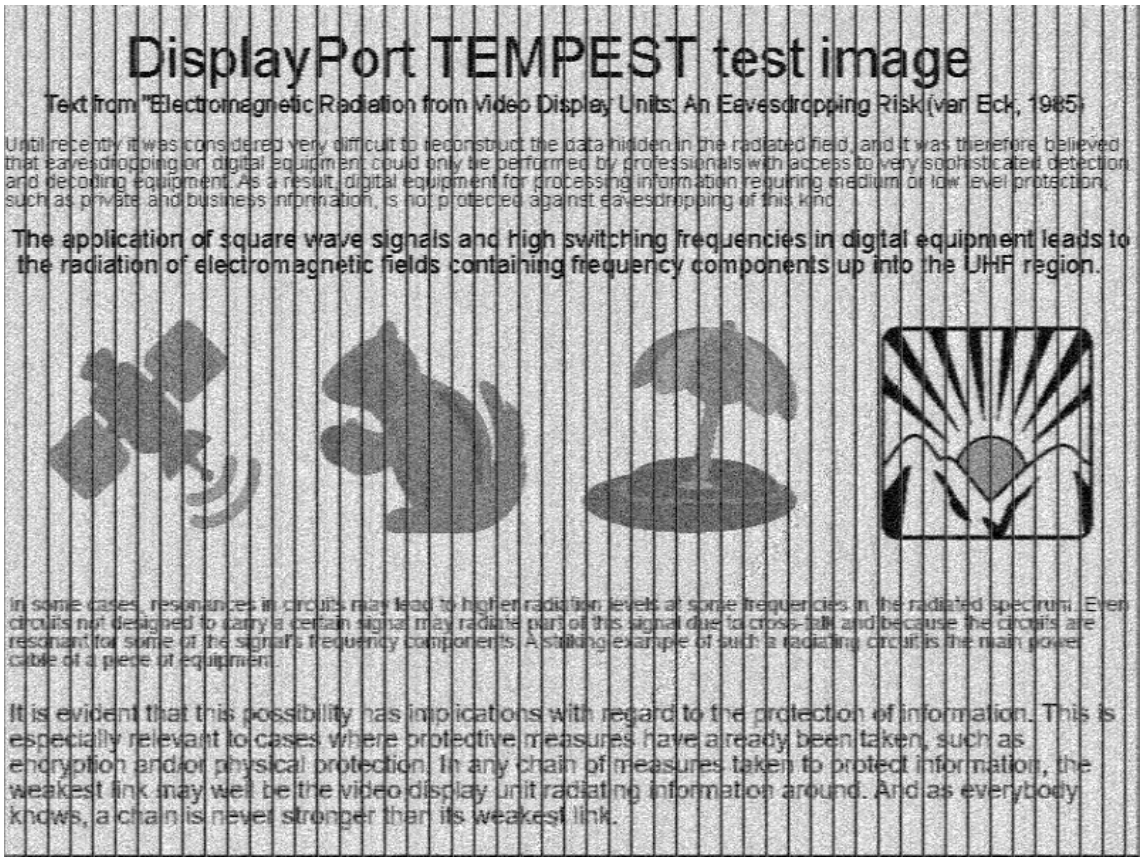


Figure 5.9: Difference between text (`#000000`) and background (`#ffffff`) channels in reconstructed image shown in Figure 5.8 (silhouette mode).

are particularly difficult to distinguish for an eavesdropper without carrier phase information. Balanced symbols in the scrambled and encoded data for such colours are encoded as complementary 10-bit symbols, and the emitted signals are thus significantly negatively correlated. If only the magnitude of correlation is available, complementary colours will seem similar, since sign information has been discarded. With this new algorithm, I not only recover the magnitude of the correlation between the received signal and colour templates, but also the sign, and can therefore distinguish the two.

Figure 5.11 shows reconstructed images for a range of antenna distances. The images were produced by averaging 30 frames, as opposed to 120 in Figure 5.8, to make differences in noise level more visible. Initial scrambler synchronisation was successful up to 9 m; at 10 m, the output only contains noise due to failed synchronisation. Note that the output noise level does not always increase with distance due to complex indoor signal propagation characteristics (e.g. reflections and scattering).

5.5.4 Eavesdropping range

Successful DisplayPort eavesdropping using techniques described in this thesis requires a received signal-to-noise ratio that is sufficient for three algorithm stages to succeed:

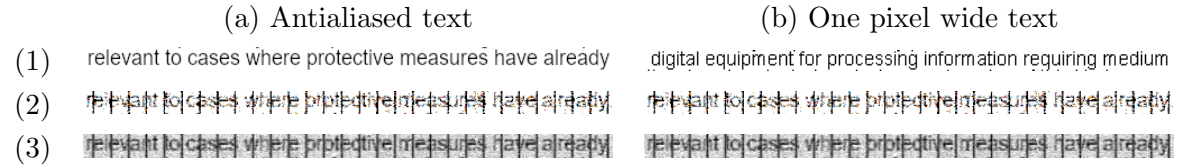
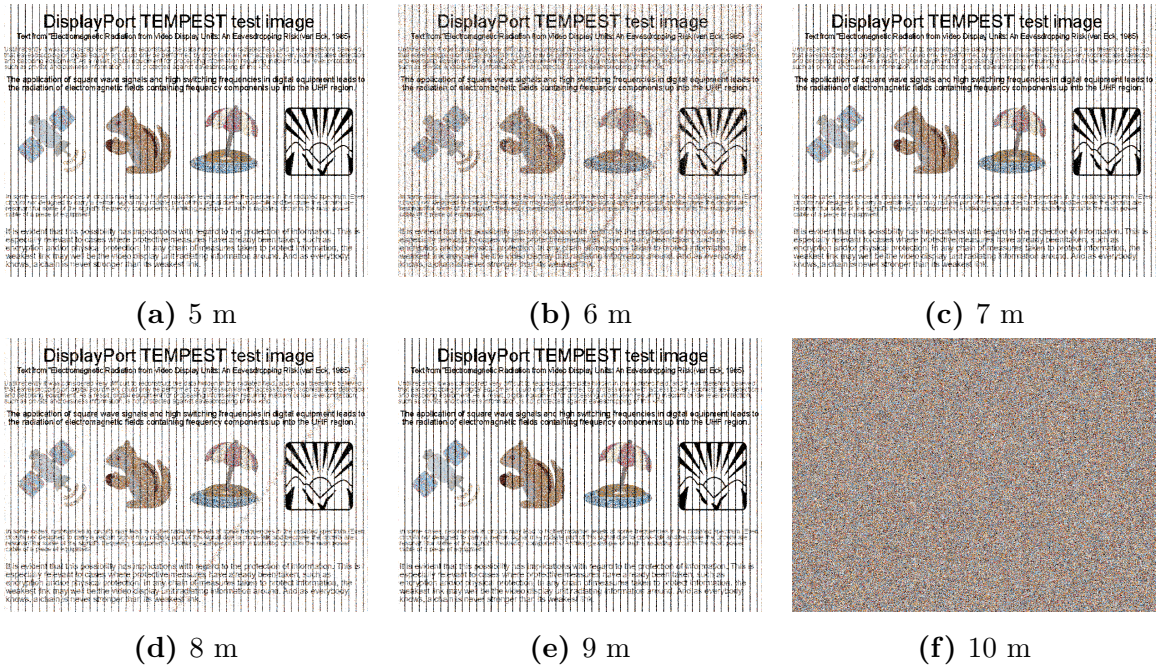


Figure 5.10: Samples of (1) original image, (2) voting mode output, and (3) silhouette mode output showing difference between foreground and background channels.



(a) 5 m **(b)** 6 m **(c)** 7 m

(d) 8 m **(e)** 9 m **(f)** 10 m

Figure 5.11: Test image reconstructed using voting-mode at varying antenna distances for 30 periodically averaged frames.

1. The scrambler timing predicted by the reset model (Section 3.3) must be approximately correct to provide an initial position estimate and reset signal for tracking.

2. The scrambler code phase tracking (Section 4.2) and carrier phase tracking (Section 5.1) loops must successfully lock onto the scrambler signal.

3. After coherent averaging, the noise level must be low enough to distinguish the candidate colours and produce an acceptable reconstructed image.

Reset model range was evaluated in Chapter 3 (Figures 3.7, 3.8). To investigate the limitations of the later algorithm stages, I used the same recordings as above, combined with a noise recording to simulate varying levels of attenuation. I made the noise recording with the target monitor off, without otherwise changing the setup, immediately before recording the dataset used for the image reconstruction demonstration.

Under free-space propagation, received power at distance d from the monitor decreases with $1/d^2$. Samples returned by the SDR are measurements of antenna voltage, which is

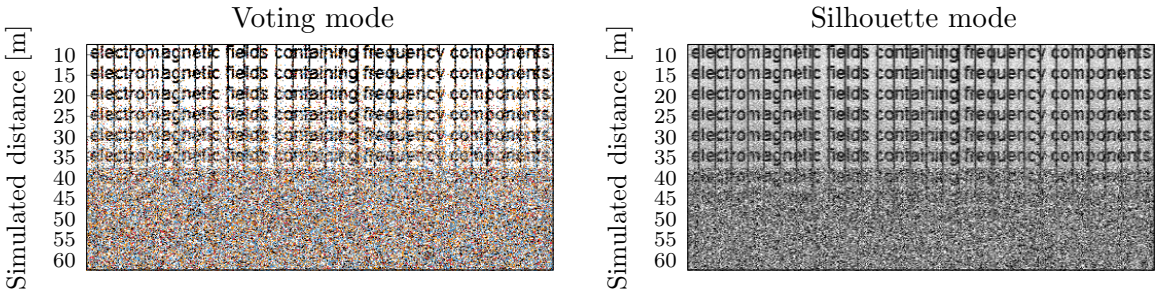


Figure 5.12: Samples of images reconstructed from 2 s long weighted combinations of received signal and recorded noise, simulating what the eavesdropper would receive at larger distances, with initial reset model parameters computed from the original 8 m antenna distance recording.

proportional to field strength at the antenna, i.e. to the root of received power. Therefore, given a recording $s[n]$ made at distance $d_0 = 8$ m and a noise recording $s_\mathrm{n}[n]$, we can compute a synthetic $s_\mathrm{s}[n]$ recording close to what the eavesdropper would record at distance $d > d_0$ as a weighted linear combination:

$$s_\mathrm{s}[n] = \frac{d_0}{d} s[n] + \left(1 - \frac{d_0}{d}\right) s_\mathrm{n}[n] \tag{5.15}$$

While indoor signal propagation differs in practice from free-space propagation, this approximation was found by O'Connell to be useful for evaluating electromagnetic eavesdropping algorithms [26]. He concluded that such simulated recordings are favourable to the eavesdropper since they do not account for additional distortion (e.g. due to multipath propagation) that would exist in a real environment at larger distances, but the two have comparable signal-to-noise ratios and signal amplitudes [26].

I used the original reset model parameters (x_0, R, P) computed from $s[n]$, and in later stages of the eavesdropping algorithm (scrambler tracking, demodulation) reconstructed an image from $s_\mathrm{s}[n]$. This allowed me to estimate the range at which the later stages could be successful if the reset model algorithm was improved. Figure 5.12 shows samples from reconstructed images for such synthetic recordings and the equivalent distance assuming uniform propagation.

Up to a simulated distance of 35 m, the image quality mainly decreases due to a lowered signal-to-noise ratio. The grayscale text image is more robust to added noise than colour classification, and remains readable up to this point. At larger simulated distances, image quality decreases sharply: the colour output is indistinguishable from noise, and the text in the grayscale image is no longer readable.

This sharp decrease happens when the signal-to-noise ratio falls below the threshold required for successful scrambler tracking. Figure 5.13 shows a plot of the average tracking error for a range of simulated distances, which increases slowly up to 35 m and becomes significantly higher at larger distances. At 50 m simulated distance, the $\pi/2$ average error

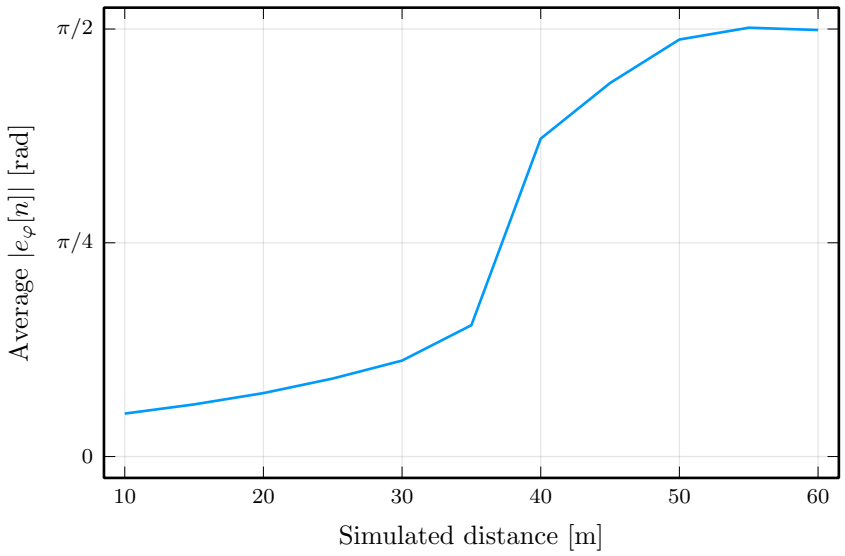


Figure 5.13: Average phase tracking error at differing simulated distances, based on a recording made at 8 m antenna distance.

is no better than random chance. At shorter distances, the noise amplitude in the reconstructed image increases due to two factors: a lower signal-to-noise ratio in the received signal and an increase in phase alignment error.

While an evaluation of eavesdropping range using simulated noisy recordings does not fully model realistic conditions, successful image reconstruction at 35 m distance provides an upper bound for the performance of the current algorithm with the setup (receiver, bandwidth, antenna, target) I used. The initial scrambler search and reset model computation fails well before scrambler tracking, at around 12 m, and so extending the range algorithmically would require improving this initial stage of the eavesdropping algorithm first.

5.5.5 Integration time

Assuming that the image shown on the monitor is stationary (e.g. a presentation slide) and that synchronisation and tracking are successful, the noise level in the reconstructed image can be reduced to produce a readable output by averaging sufficiently many frames. The tracked scrambler position $i_s[n]$ allows the eavesdropper to align recorded frames over a long time period. Further, the 512-line reset period can be used to align recordings made several minutes apart, since the resulting images can be vertically aligned by calculating the number of such periods between the recordings.

Figure 5.14 shows samples of the image reconstructed from a contiguous 2 s recording for varying averaged frame counts, up to the 120 frames spanning the entire recording (equivalent to Figure 5.8).

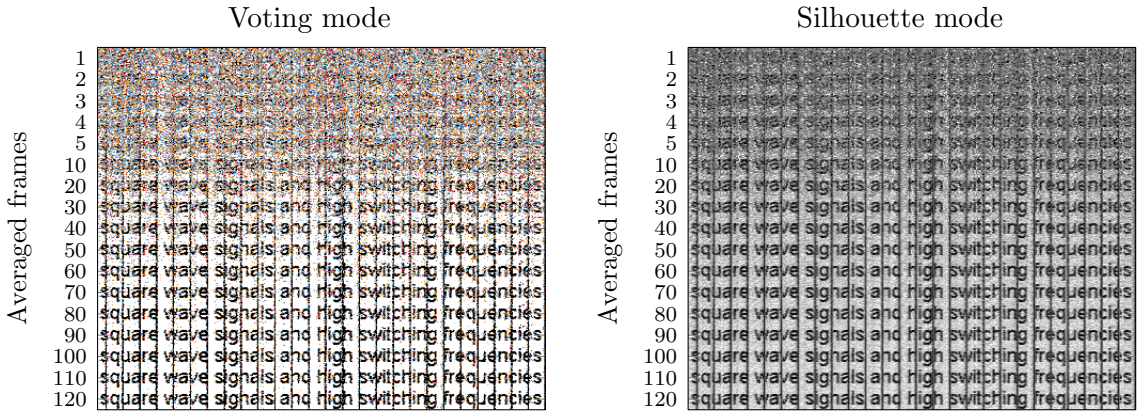


Figure 5.14: Samples from an image reconstructed from recording made at 8 m antenna distance, for a varying number of coherently averaged frames.

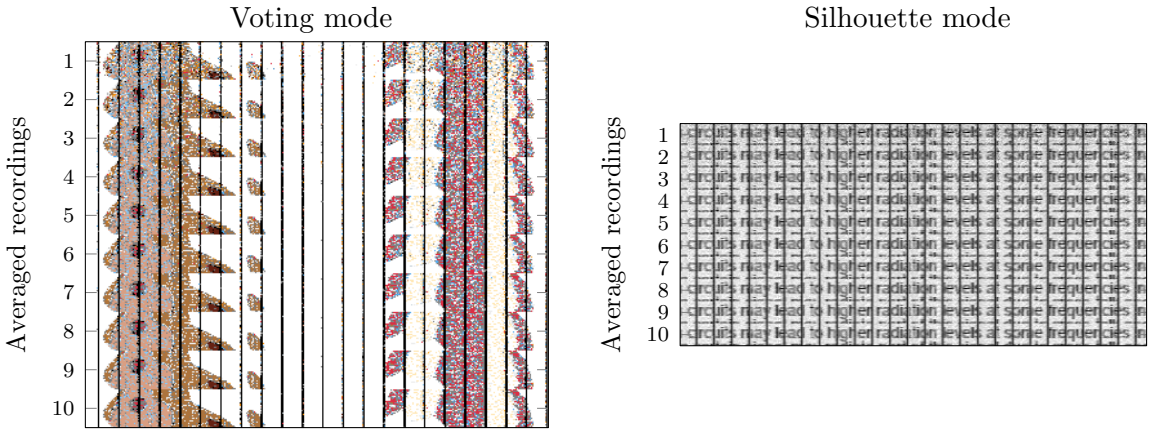


Figure 5.15: Samples from an image reconstructed by combining multiple outputs from two-second recordings made at 8 m antenna distance.

To demonstrate the feasibility of long-term averaging, I made ten recordings, each 2 s long, and for the i-th recording I stored the starting time t_i, as measured by the software-defined radio receiver clock. The time between the first and last recording was $t_{10} - t_1 = 130$ s, and the average time between them was $130/9 \approx 14.5$ s.

For each recording i, I separately computed the reset model parameters $\boldsymbol{\theta}_i = (x_{0i}, R_i, P_i)$. The number of image lines l_i between x_{01} and x_{0i} can be estimated by the number of 512-line reset periods as

$$l_i = 512 \cdot \left\lfloor \frac{x_{0i} - x_{01}}{R_i} + 0.5 \right\rfloor \tag{5.16}$$

and the number of image lines between the start of the recording (i.e. the first image line in the output) and x_{0i} as $k_i = \lfloor x_{0i}/(R/512) + 0.5 \rfloor$.

The image outputs for all recordings were already horizontally aligned, since the eavesdropping algorithm operates on entire lines. I vertically aligned the colour scores $\bar{u}_k[n]$ for each recording by vertically shifting them by $l_i + k_1 - k_i$. Samples from the resulting

images are shown in Figure 5.15. The improvement is small for silhouette mode, but noticeable for voting mode.

This vertical alignment is possible even for a very long delay between recordings. Let R be the average reset period over the time period between two recordings, R_2 the period estimated by the E/M scrambler synchronisation algorithm, and Δx the time between recordings in samples. The recordings will be misaligned if

$$\left|\frac{\Delta x}{R} - \frac{\Delta x}{R_2}\right| > 0.5 \tag{5.17}$$

or, writing $\Delta x = l \cdot R/512$ in terms of the number of lines l between the recordings and rearranging:

$$\left|1 - \frac{R}{R_2}\right| > \frac{256}{l} \tag{5.18}$$

Assuming that $R \approx R_2$, the left-hand side is approximately equal to the relative error $\delta_R = |R - R_2| / R$, and we can estimate the maximal distance in lines as

$$l > \frac{256}{\delta_R} \tag{5.19}$$

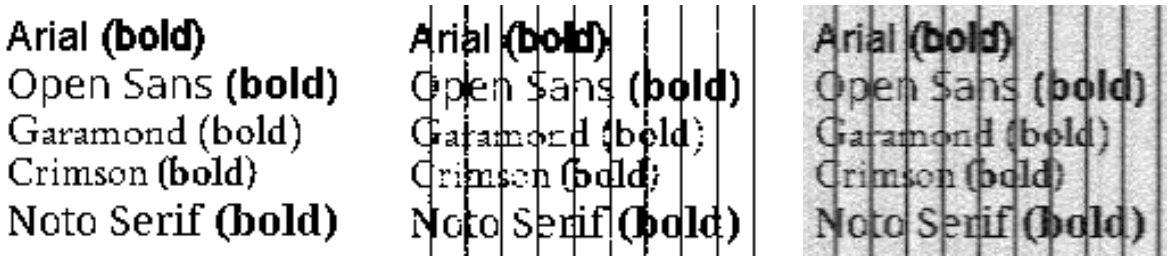
A very pessimistic $\delta_R = 30$ ppm error (i.e. if the reset model is no better than estimating R from the nominal bitrate) gives $l = 8.5 \cdot 10^6$ frames, meaning that for the 800 × 600 @ 60.3 fps video mode, vertical alignment is still possible for recordings made $\frac{l}{h_\mathrm{t} f_\mathrm{v}} = \frac{l}{628 \cdot 60.3\ \mathrm{Hz}} = 225$ s apart.

5.5.6 Image characteristics

In this section, I present several cropped image samples that show the behaviour of the image reconstruction algorithm when applied to images, with emphasis on different characteristics: various fonts, the use of antialiasing, textured backgrounds, colours that are particularly different to distinguish, and combinations of static text and video. All recordings featured in this section were made at 2 m distance, with the same recording setup as before. This short eavesdropping distance allows us to more clearly see the impact of these features and identify limitations present even in recordings with a high signal-to-noise ratio.

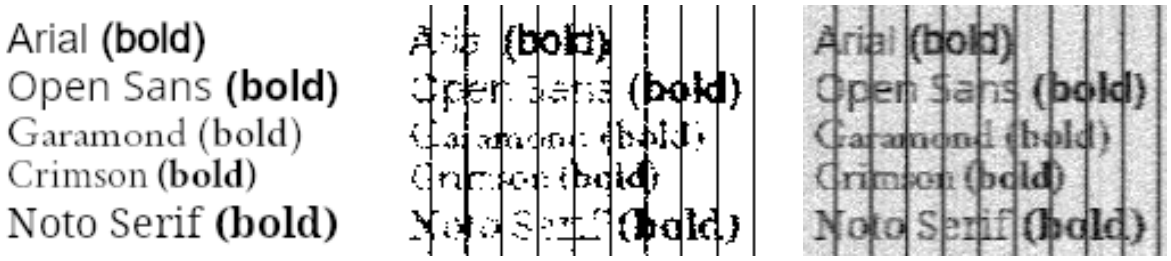
Fonts and antialiasing

Figure 5.16a shows a cropped sample of the displayed image, which contains two sans-serif typefaces (Arial and Open Sans) and three serif typefaces (Garamond, Crimson, and Noto Serif), rendered with antialiasing disabled at 16 pixel per em size. Without antialiasing, the image only contains two colours: white (`#ffffff`) background and black (`#000000`) foreground.



(a) **(b)** **(c)**

Figure 5.16: (a) Displayed image featuring five fonts rendered with antialiasing disabled, at 16 pixel/em size, and (b) voting and (c) silhouette-mode reconstructions from 2 s long emissions recorded at 2 m distance.



(a) **(b)** **(c)**

Figure 5.17: (a) Displayed image featuring five fonts rendered with antialiasing enabled, at 16 pixel/em size, and (b) voting and (c) silhouette mode images reconstructed from 2 s long emissions recorded at 2 m distance.

The reconstructed text in both voting-mode (Figure 5.16b) and silhouette-mode (Figure 5.16c) images is readable, regardless of the font. The main difference in readability is not due to the choice between serif and sans-serif, but rather due to thin vertical strokes in Garamond and Crimson, and to a lesser extent in Open Sans.

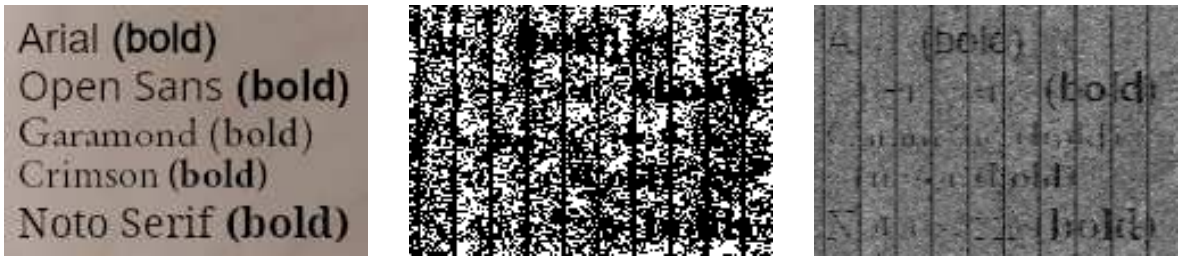
If the text is rendered with antialiasing enabled (Figure 5.17a), voting-mode reconstructed text (Figure 5.17b) becomes nearly unreadable, since most pixels in a letter no longer match the assumed foreground colour. Only boldface text in fonts featuring wider strokes, in particular Arial, remains legible, since it still contains black pixels after anti-aliasing. When the image is reconstructed in silhouette mode, however, antialiased text can be clearly seen, regardless of font, as regions that do not match the background colour (Figure 5.17c). This was also demonstrated for the initial attack in Section 3.5.4.

Textured background

Since antialiased text can still be recovered from the background channel, one possible countermeasure might be to replace the single-colour background with a textured image. Figure 5.18 shows one such image and corresponding reconstructed outputs, using a photograph of a desk surface as the background behind black text rendered with antialiasing

(a) **(b)** **(c)**

Figure 5.18: (a) Displayed image with a textured background and text as in Figure 5.16, and (b) voting and (c) silhouette-mode reconstructions from 2 s long emissions recorded at 2 m distance.



(a) **(b)** **(c)**

Figure 5.19: (a) Displayed image with a textured background and text as Figure 5.17, and (b) voting and (c) silhouette-mode reconstructions from 2 s long emissions recorded at 2 m distance.

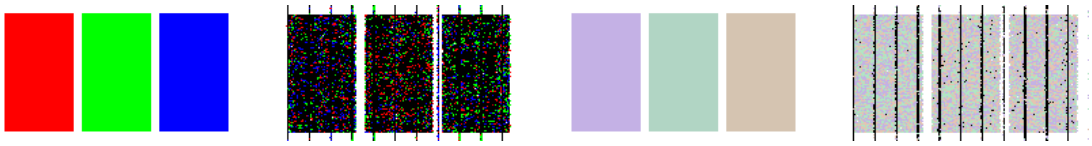
disabled. A second example, with antialiasing enabled, can be seen in Figure 5.19.

We can see that a textured background alone is not a sufficient mitigation against eavesdropping, but can be in combination with antialiased text, particularly for non-bold fonts where few pixels are rendered as black and thus visible in the foreground channel.

Cyclically shifted colour channels

One limitation of the attack described in Chapter 3 was the inability to distinguish between cyclically shifted colour channels, since scrambler alignment was not accurate enough to correctly assign channels to transferred bytes when constructing templates. Even though scrambler code tracking allows us to accurately align the received signal with the scrambler replica, the algorithm presented in this chapter suffers from the same limitation: this channel assignment requires not only accurate scrambler tracking, but also a byte-accurate model of DisplayPort data framing. As discussed in Section 4.1, I have successfully reverse-engineered the algorithm used for line padding in the targeted Intel video interfaces, but not the algorithm used to insert leap bytes.

Figure 5.20 shows the effect of this limitation in practice, with two sample sub-images containing cyclically shifted colours: in the first, cyclic shifts of $(\texttt{0x00}, \texttt{0x00}, \texttt{0x00})$, and



(a) (b) (c) (d)

Figure 5.20: (a), (c): test patterns featuring cyclically shifted RGB colours, and (b), (d): voting-mode reconstructions from 2 s long emissions captured at 2 m distance. Pixels are assigned the highest-scoring colour from five candidates: black, white, and the three colours in the displayed image, namely (a), (b): cyclic shifts of (`0x00`, `0x00`, `0x00`), and (c), (d): cyclic shifts of (`0xca`, `0xb1`, `0xe5`).

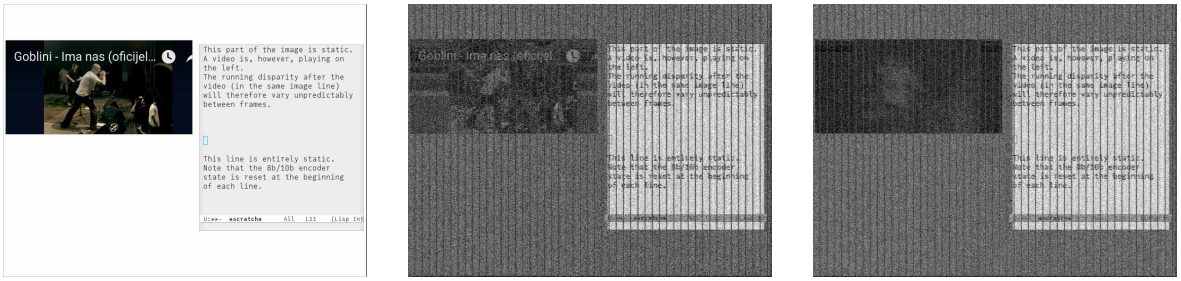
in the second, cyclic shifts of (`0xca`, `0xb1`, `0xe5`). In both voting-mode reconstructed images, each pixel was assigned the highest-scoring colour from five candidates: black, white, and the three colours present in the displayed image. In both cases, the resulting colour regions are indistinguishable and contain an even mixture of all three candidate colours. The three pure RGB colours are also frequently misclassified as black, since the data will match the black template in two bytes out of three for each pixel, as opposed to $(3 + 1 + 1)/3 = 1.67$ matches on average for random channel alignment.

Video

All image reconstruction algorithms in this thesis assume that the image is static, and that channel scores can, therefore, be periodically averaged at the frame rate to improve the signal-to-noise ratio. Image regions with changing content, such as video players, are therefore not suitable targets. One might expect that this would also affect image reconstruction on the right of the changing region, as a changing pixel causes the encoder state to unpredictably invert between frames and the change propagates until the end of the line.

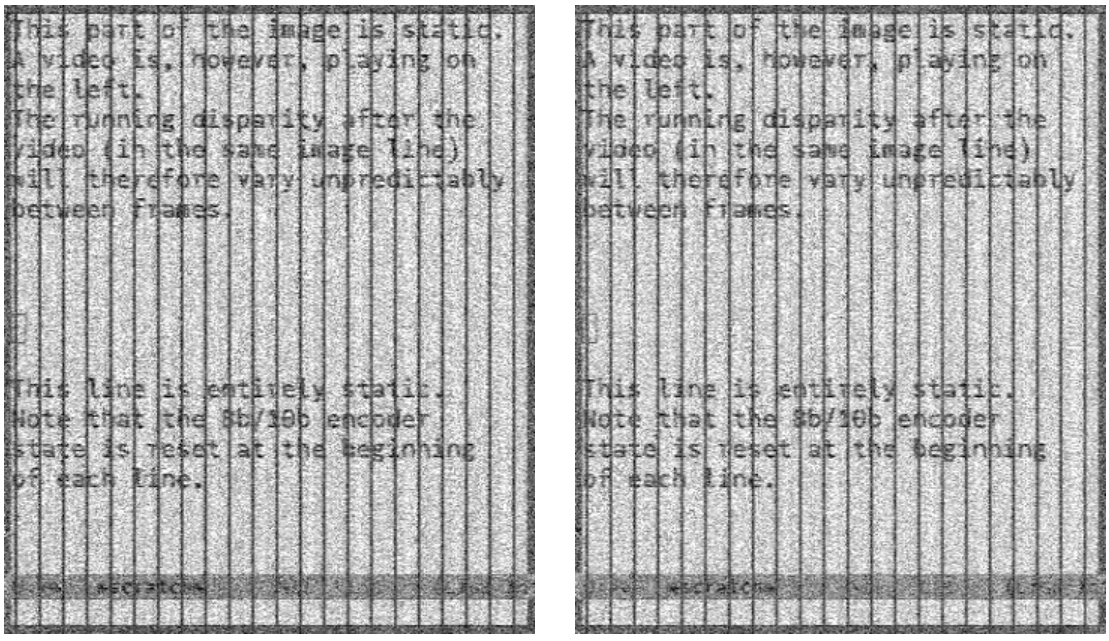
However, since templates used in the coherent demodulation algorithm only include balanced symbols, the resulting correlation is unaffected by changes in the running disparity. Therefore, any static region in the displayed image can be reconstructed, regardless of changes in the rest of the image.

As a demonstration, consider the screenshot in Figure 5.21a containing a (paused) video player and a text editor. Figure 5.21b shows the reconstructed silhouette-mode image with the video paused, and Figure 5.21c the same for a recording made while the video was playing. As expected, the static text editor is unaffected by the player, and there is no observable difference between lines that also contain a video on the left of the editor and those that do not. The text-editor window can be seen in more detail in Figure 5.22.



(a) **(b)** **(c)**

Figure 5.21: (a) Screenshot of monitor showing a video-player and text-editor window, (b) silhouette-mode image reconstructed from a 2 s long recording made at 2 m distance with the video paused, and (c) reconstructed image with the same parameters and the video playing. Reconstructed images show the difference between channels corresponding to the editor background (`#ededed`) and foreground (`#2e3436`) colours.



(a) **(b)**

Figure 5.22: Cropped samples of reconstructed images shown in Figure 5.21, with the video (a) paused and (b) playing.

(a)

(b)

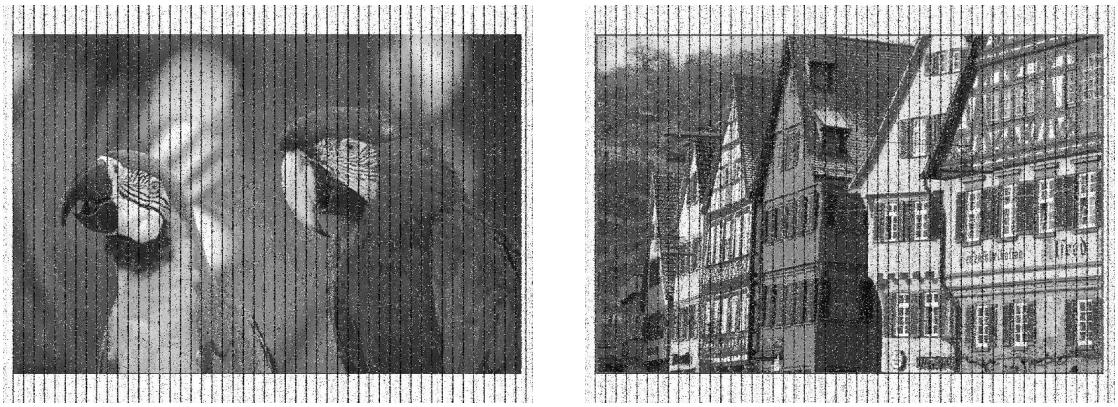
Figure 5.23: Grayscale 768 × 512 test images, created by converting images from the Kodak Lossless True Color Image Suite [81] to grayscale without rescaling.

5.5.7 256-colour grayscale image

Voting-mode image reconstruction can be used to produce not only readable text, but also continuous-tone photographs, if the set of candidate colours is small enough that constructing templates for each colour is computationally feasible. Here, I demonstrate that a recogniseable image can be reconstructed from the emissions of a monitor showing a grayscale photograph, with 256 candidate colours, captured at 2 m distance. This shows that unlike eavesdropping on HDMI interfaces using amplitude or phase demodulation, which results in false-colour images where visually similar colours yield unrelated demodulation outputs, template-based DisplayPort eavesdropping can target photographic images.

The displayed images, shown in Figure 5.23, were created by converting photos from the Kodak Lossless True Color Image Suite [81] to grayscale. They were displayed without rescaling, with the original 768 × 512 dimensions. Figure 5.24 shows the result of voting-mode reconstruction, where each pixel is assigned the highest-correlation grayscale colour out of 256 possible values.

This is, however, a computationally expensive demonstration. With my implementation of the attack, processing each of the 2 s long recordings took approximately 90 minutes without sub-template optimisations described in Section 4.3. These optimisations would reduce the number of required templates from 256 to 61, and since the computational cost is dominated by template construction, the processing time would decrease to around 20 minutes. This is still far from a practical real-time attack (although an eavesdropper can make recordings to be processed at a later time, like this demonstration). Significant improvements in efficiency would likely require either algorithmic improvements, or an implementation leveraging e.g. a highly parallel GPU rather than a four-core CPU.



(a) **(b)**

Figure 5.24: Grayscale photographs reconstructed a 2 s long recording of emanations made at 2 m distance. Each pixel is assigned the highest-scoring colour from 256 candidate colours (voting mode).

5.6 Extensions

Finally, let us look at two extensions to the eavesdropping algorithm that are not directly related to image quality: an optimisation for black-and-white images, and automated vertical alignment that assumes the bug present in the tested Intel display controllers.

5.6.1 Black-and-white images

Consider a situation where the eavesdropper is only interested in a grayscale image showing the difference between a pair of complementary colours, such as white ($c_1 =$ `#ffffff`) and black($c_2 =$ `#000000`). The output image is created by rasterizing the perodically averaged score difference $\bar{u}_1[i] - \bar{u}_2[i]$. Before periodic averaging, the scores are computed as (Equation 5.13)

$$u_k[n] = s[n]\tilde{z}_k[n_t]^* \tag{5.20}$$

and since the two colours are complementary $\tilde{z}_1[n_t] = -\tilde{z}_2[n_t]$. Therefore, $\bar{u}_1[i] - \bar{u}_2[i] = 2\bar{u}_1[i]$, and it is enough to synthesise only one of the templates and rasterize the resulting score $\bar{u}_1[i]$.

5.6.2 Automated vertical alignment

While analysing recordings of DisplayPort data, I discovered that, for the Intel video interfaces I examined, blanking start symbols are inserted six bytes earlier in the vertical blanking period than at the end of an image data line (Section 3.1.5). If one of these is replaced by a scrambler reset, the entire scrambling sequence in the following 512 lines will be shifted by the same offset.

We can detect the reset periods in which this happens by measuring the error of scrambler positions predicted by the reset model without tracking, with the slow error function e_s used in the acquisition phase of scrambler tracking (Section 4.2.2). In periods that start in the vertical blanking period, the e_s is six bytes lower than in other periods, and therefore always negative, unlike small positive errors in other periods.

Detecting a single such reset period provides information about the vertical position of the line at which it starts, since it is guaranteed to be in the vertical blanking period, i.e. between h_d and h_t. This is enough information for useful vertical alignment: if the image is vertically shifted such that this line is the bottom-most, the image contents will always be contiguous and will not wrap around the bottom. The remaining blanking lines will appear at the top of the image if the eavesdropper cannot find the first blanking line. However, even such partial alignment can be helpful.

The probability that a randomly-chosen reset period starts in the blanking region is $p = \frac{h_\mathrm{t}-h_\mathrm{d}}{h_\mathrm{t}}$. Assuming for simplicity that this choice is made independently for each period, the expected number of periods before observing one that is useful for vertical alignment is p^{-1}, or $512p^{-1}/h_\mathrm{t} = \frac{512}{h_\mathrm{t}-h_\mathrm{d}}$ frames. For example, for the 1920×1200 @ 59.95 fps and 800×600 @ 60.3 fps video modes used in demonstrations in this thesis, the expected number of frames needed would be 15 and 20 respectively.

Exact alignment requires a reset to happen in the first line of the vertical blanking period, which can take significantly longer. If $\gcd(512, h_\mathrm{t}) = 1$, the line positions at which resets occur cycle through all lines in the image before repeating, and so the expected number of periods before seeing any particular position is $h_\mathrm{t}/2$. Each period is 512 lines long, and so the expected number of frames is $\frac{512h_\mathrm{t}/2}{h_\mathrm{t}} = 256$, or around 4.3 s (twice as long, 8.5 s, to guarantee success). If the total height h_t is divisible 2, the scrambler will never reset on an odd-numbered line, halving the expected time (or reducing it further if h_t is divisible by a larger power of 2).

Chapter 6

Concluding remarks

The main contribution of this thesis was to demonstrate that electromagnetic eavesdropping on DisplayPort interfaces is possible. Scrambled and encoded signals cannot be eavesdropped on using amplitude demodulation receivers or other techniques that have been previously described in the literature, and have been considered a difficult and possibly infeasible target. I have shown that a short, fixed scrambling sequence is not sufficient to protect information, and that template-based algorithms can be used to classify image pixels based on colour.

In Chapter 3, I presented an overview of DisplayPort, focusing in particular on data framing, scrambling, and encoding. This overview is primarily based on the protocol specification, but also includes reverse-engineered details of the implementation-specific padding algorithm used by the interfaces I examined. I then described a two-part eavesdropping attack, in which I first estimate the target's scrambler timing structure using a three-parameter reset model, which I then use to classify transmitted pixels based on the received signal's correlation with templates for a set of candidate colours. I demonstrated that the reset model parameters can be successfully estimated at up to 8 m to 12 m distance, depending on the target video mode, and that the image reconstruction algorithm can recover readable text at 5 m distance.

Next, in Chapter 4, I extended the scrambler synchronisation algorithm with a code tracking delay-locked loop based on a PID controller. This code tracking approach, similarly to a GPS receiver, uses two discriminators to estimate the code phase error: the first computes the correlation of the received and expected signals at many points, while the second quickly estimates the error from only four such correlations. The output of the discriminators is used to update the current estimate of the target link bitrate and keep the local scrambler replica synchronised with the target. I then described three properties of the 8b/10b encoding used in DisplayPort, which enabled me to describe correlations between the received signal and templates for all 2^{24} RGB triples using only 61 sub-templates. I showed that this efficient representation, together with code phase tracking, can be used

to enumerate all colours in an image region of interest and identify the background colour of the image without any prior knowledge.

In Chapter 5, I extended the scrambler tracking algorithm to also track the carrier phase using a phase-locked loop, which eliminates the previously arbitrary phase offset between the received signal and colour templates. This enabled coherent averaging of the complex-valued colour scores, which allows the eavesdropper to distinguish complementary colours based on their sign, and also increases the output signal-to-noise ratio. This increased signal-to-noise ratio allowed me to also improve the horizontal resolution in the reconstructed image by eliminating a correlation window used in the initial algorithm in Chapter 3. I demonstrated that, using carrier code and phase tracking together with coherent averaging, I can recover clearly readable text from 8 m distance under realistic conditions. This distance was primarily limited by initial timing parameter estimation, and I showed that later parts of the image reconstruction algorithm succeed at up to 35 m simulated distance.

The work described in previous chapters has all been from the perspective of an attacker. Finally, before suggesting directions for future research on the topic, I will briefly discuss the same problem from a defender's point of view and suggest some possible countermeasures that would make eavesdropping more difficult.

6.1 Countermeasures

Electromagnetic eavesdropping could be made significantly harder by modifying the DisplayPort protocol, for example such that it uses a much longer and variable scrambling sequence with parameters chosen randomly when the connection is established. However, such a redesign of an existing standard to mitigate a threat that only affects a very small fraction of its users is hardly practical. Even if it was possible, substantial changes to the protocol would not be compatible with the many existing DisplayPort devices.

I will therefore limit my discussion of possible countermeasures to those that do not require protocol-level changes. These include measurement standards used to certify devices as secure against electromagnetic eavesdropping, and software countermeasures that do not require any hardware modifications.

6.1.1 Hardware emission standards

TEMPEST certification for government and military use is based on a zone model. The relevant NATO standards, SDIP-27/28/29, are classified, but publicly available documents state that locations are classified into zones based on the distance from a potential eavesdropper [64]. A zone 0 classification implies that an attacker can come very close

to the location where sensitive data is processed, and zones 1 and 2 mean that no attacker can be closer than 20 m or 150 m respectively. Equipment is certified to levels matching these zones [82] if it passes the relevant compliance tests. Details of the testing procedure are classified, but some information is available in unclassified documents: radiated power limits are significantly lower than civilian EMC limit curves [63], and unlike EMC measurements, TEMPEST labs evaluate if signals are correlated with processed data [83], using data designed to produce specific signal characteristics [84] such as images containing vertical black and white stripes.

While radiated power measurements are a useful tool for estimating the signal power available to an eavesdropper, limit curves designed for non-scrambled interfaces might be too lax to ensure that DisplayPort devices are secure. One of the main motivations for scrambling data transmitted over a high-speed interface is to reduce peak levels in the radiated power spectrum by spreading power that would otherwise be contained in a narrow frequency range (e.g. a square wave for a bitstream with alternating zeroes and ones) across the spectrum. This is a useful mechanism for meeting commercial EMC regulations, since strong emissions at a single frequency are more likely to interfere with nearby receivers than weak emissions across a broader frequency range with the same combined power. The same argument does not necessarily hold for security, since an attacker with a sufficiently broadband receiver can still receive a substantial part of the unintentionally radiated scrambled signal. The designer of limit curves for TEMPEST-certified equipment which uses scrambling should therefore consider imposing stricter limits than those for earlier video interfaces.

A certification lab can use specially designed displayed images which undo the scrambling to estimate the total signal power available to an eavesdropper using a narrowband measurement. Let $p[x, y]$ be the RGB pixel value in column x and row y, and let $\Xi[n]$ be the n-th scrambler byte applied in the image region after a reset, skipping the scrambling sequence used in blanking periods and fill regions. An image that undoes the scrambling in every line following a reset can be designed, for example, by setting

$$p[x, y] = (\Xi[3 \cdot x] \oplus \texttt{0x4a}, \Xi[3 \cdot x + 1] \oplus \texttt{0x4a}, \Xi[3 \cdot x + 2] \oplus \texttt{0x4a}) \tag{6.1}$$

such that all scrambled image bytes will be `0x4a`, which is 8b/10b encoded as alternating zeroes and ones. The transmitted bitstream will then contain a large frequency component at $f_\mathrm{b}/2$ every 512 lines, which will be visible as a single peak in the measured power spectrum. A longer identifiable signal could be produced by choosing pixel data in subsequent rows to undo later parts of the scrambling sequence, at the cost of a longer wait before a scrambler reset happens at a line where this pattern begins. In either case, the measurement procedure must allow enough time for such infrequent signals to be identified.

The part of the scrambling sequence used for image bytes $b[n]$ depends on padding implementation specifics used by the device being examined. This means that rather than

using a single test image, it would have to be tailored to the implementation used by the device under test. It might be helpful to maintain a collection of test images, and to require that manufacturers seeking TEMPEST certification disclose the specifics of their padding algorithm.

Autocorrelation can be used to test that a signal possibly contains compromising emanations from a non-scrambled interface such as DVI and HDMI. Since the transmitted data is periodic at the frame period, and also to a lesser extent at the line period, autocorrelation peaks at these positions suggest that a signal contains image-related data. This does not hold for DisplayPort, since scrambled data is not frame-periodic. An equivalent correlation test would have to search for correlations at a significantly longer delay T at which the scrambling sequence repeats and is aligned with image data at the same position. The period $T = kf_{\mathrm{v}}^{-1}$ is then a multiple of the frame period such that the scrambler resets in the same image line k frames apart. Since resets happen every 512 lines, significant correlation is expected at $T = 512f_{\mathrm{v}}^{-1} \approx 8.53$ s (for $f_{\mathrm{v}} = 60$ Hz) for all video modes. Significant correlation at smaller multiples of the frame period is only present in some video modes, where the padded image height is even (i.e. shares a factor with 512).

Finally, the test might include computing a cross-correlation between the received signal and a synthesised scrambling sequence similar to that used to identify scrambler reset times in Section 3.3.1. This should be done while the target monitor displays an all-black image to maximise the number of resets that can be identified and emulate the best-case scenario of an attacker.

6.1.2 Software countermeasures

Several software countermeasures that reduce the effectiveness of electromagnetic eavesdropping have been proposed for non-scrambled video interfaces. These include fonts designed to be difficult to read in an eavesdropped image [45–47], colour combinations that look similar after demodulation [3], and partial randomisation of image contents [3].

Both types of proposed TEMPEST fonts are designed specifically for analog interfaces, which leak mainly high-frequency components of the displayed image, and demodulation behaves similarly to an edge detector. The resulting text can be made harder to read either by lowpass filtering the glyphs to eliminate said high-frequency components [45, 46], or by making the outlines similar and therefore difficult to distinguish in an image only containing edges [47]. Both of these techniques are of limited use for digital interfaces, and are not applicable at all for DisplayPort. Scrambling spreads emitted power across the spectrum, and so filtering displayed glyphs has no effect on the emitted spectrum. Additionally, the attacks described in this thesis are based on pixel classification rather than edge detection.

In digital interfaces such as HDMI, certain colour pairs produce similar images after demodulation because their TMDS-encoded representations are similar, even though the colours are visually distinct [3]. Due to scrambling and 8b/10b encoding, such low-contrast pairs only partially exist for DisplayPort. A pair of colours that share the same five least-significant bits will have identical balanced 6b encoded blocks, but will differ unpredictably due to scrambling in the top four bits, and will therefore reduce but not eliminate the available signal-to-noise ratio. Circularly shifted colours can also be effective against an attacker who cannot exactly align templates with the received signal, since a RGB colour (r, g, b) will match a template for (g, b, r) that is misaligned by one byte. Eliminating such misalignment requires that the attacker can both track the scrambler accurately and predict the exact padding in blanking regions, including leap bytes.

Information from the lower 6b encoder block can also be eliminated by randomising five low bits in displayed colours. This can be an effective countermeasure for TMDS-encoded pixel data, where changes in the low bits can affect the entire encoded symbol [3]. Since the 8b/10b encoder used in DisplayPort encodes low and high bits separately, randomising low bits does not affect high bits other than possibly inverting the running disparity. A randomisation method that scrambles both low and high bits would at a minimum have to be applied to the lower six bits of each byte and therefore produce a visually noisy image.

If possible, randomising padding bytes in fill and blanking regions can serve as a countermeasure against synchronisation and tracking algorithms described in this thesis, which exploit the known region of zero bytes to align the received signal with a scrambler replica. DisplayPort allows an interface to replace padding bytes by secondary data packets delimited with "secondary start" and "secondary end" control characters. This is commonly used to transport audio, video metadata, or vendor-specific information [85]. A modified video driver could be used to replace most of the padding with random secondary packets and eliminate known zero-valued regions.

The most effective digital countermeasure is encryption: instead of trying to minimise the amount of information about transferred data an attacker can extract from unintended emissions, encrypting this data will ensure that this information cannot be used to recover an image. Modern video standards, including DisplayPort, support encrypted video streams using *High-bandwidth Digital Content Protection (HDCP)*. This is an encryption layer designed for use as a copy protection mechanism. Only devices certified by Digital Content Protection LLC, the organisation which manages the HDCP ecosystem, are issued keys to ensure that an unauthorised device such as a capture card cannot receive HDCP-encrypted video.

The first version of the HDCP standard was a custom cipher, designed to be implementable using fewer than 10000 gates. A 2001 cryptanalytic attack targeted a weakness in the authentication protocol, showing that extracting 40 keypairs from different devices

would allow an attacker to compute the authority's master key and produce forged key-pairs [86]. The attack was demonstrated practically in 2011 [87], and the master key was separately leaked online in 2010 [88]. The newer HDCP 2 protocol uses standard primitives (RSA, AES-CTR, HMAC-SHA256) and a key establishment step similar to TLS [89]. The only published weakness targets the key exchange, and has been addressed by newer revisions of the standard. Devices that claim to remove or downgrade HDCP 2 encryption are available online, but the vendors do not provide information on whether they rely on leaked keys or protocol vulnerabilities [90].

Even weak encryption would be enough to prevent an eavesdropper from extracting image data from the unintentional emissions of an interface transmitting encrypted data. The eavesdropper in this scenario can at most see the encrypted video stream, and cannot modify the transmitted data in any way or see the key exchange (which only happens once, when the display is powered on). In practice, the leaked signal only contains a noisy and distorted representation of the data, and the attacker is weaker than even a passive man-in-the-middle observer. Even HDCP 1 should be secure with this noise-limited passive eavesdropper threat model.

6.2 Alternative designs

Eavesdropping algorithms described in the previous chapters have been specifically designed for the scrambling and encoding used in DisplayPort. In this section, I will consider other possible encoder and scrambler choices, as well as the order in which they are layered. I will focus on choices that are used in either video interfaces or other existing high-speed serial interfaces. Table 6.1 summarises these options, together with an estimate of their vulnerability to electromagnetic eavesdropping using either techniques similar to earlier TEMPEST research, or the template-based approaches that DisplayPort is vulnerable to.

Displays are an attractive electromagnetic eavesdropping target because the image contents are highly redundant (e.g. one letter consists of many pixels), meaning that demodulation can tolerate many errors or signal distortion and still produce readable text. The data is also frame-periodic for static images, which allows the eavesdropper to improve the signal-to-noise ratio by periodic averaging. Estimating the vulnerability of other protocols to eavesdropping is more difficult, and depends on the attack goal: for example, a template-based attempt to identify USB keyboard packets (and therefore keystroke timing) would be more tolerant to noise and errors than one that tries to identify the key that was pressed. Additionally, common-mode emission-reduction techniques, such as galvanic isolation and common-mode chokes used in twisted-pair Ethernet (Section 2.2.4) reduce the signal power available to an eavesdropper when compared to DisplayPort.

I therefore limit the discussion in this section to the impact of protocol design choices. In Table 6.2, I list the choices made in several commonly used protocols and label them

Scrambler	Order	Encoder		
		TMDS	block	scrambler
synchronous	scramble first	✓	✓†	
self-synchronising	scramble first			
synchronous	encode first	✓⋆	✓⋆	
self-synchronising	encode first			

Table 6.1: Summary of possible scramblers, encoders, and their layering, and an estimate of their possible vulnerability to electromagnetic eavesdropping. Block encoders include 8b/10b and other fixed-codebook codes. Scrambler encoders include 64b/66b, 128b/130b, and similar codes based on a self-synchronising scrambler. ✓: likely vulnerable to template-based attacks. ⋆: possibly vulnerable to earlier TEMPEST attacks after scrambler synchronisation and despreading. †: the DisplayPort protocol.

based on theoretical susceptibility to attacks similar to those targetting DisplayPort. Establishing practical limits, due to e.g. signal power, available integration or averaging time, and tolerance to errors, will require in-depth protocol-specific research.

6.2.1 Encoding

Fixed-codebook block codes

The 8b/10b code used in DisplayPort is one of several balanced block codes using a fixed encoding table. It is the most commonly used such code, for example in USB 3.0, Thunderbolt 1 and 2, and 1000BASE-X Gigabit Ethernet. Another example of such a code is 8b/12b, used in the low-cost, low-latency IEEE 1335 communications standard.

Since the 8b/10b encoder state consists of only one bit, scrambler synchronisation algorithms, such as discriminators in Section 4.2.2, only need two correlations to identify the scrambler state during a blanking period. Additionally, approximately 54% bits in 8b/10b-encoded data are independent of the state, which allows the eavesdropper to only include those balanced symbols in templates.

Transition Minimised Differential Signaling (TMDS)

A TMDS encoder maps also bytes to 10-bit symbols. Data bits are first transformed to reduce the number of transitions by either replacing each data bit with its XOR with the previous encoded bit, or by the XNOR. The choice of transformation is indicated by the ninth bit. Finally, data bits may be inverted to maintain balance, as indicated by

Protocol	Encoder	Scrambler	Attack
DisplayPort	8b/10b	synchronous	✓
DisplayPort 2.0, < 10 Gbit/s	8b/10b	synchronous	✓
DisplayPort 2.0, ≥ 10 Gbit/s	128b/132b	self-synchronising	
USB 3.0	8b/10b	synchronous	✓
USB 3.1 (≥ 10 Gbit/s)	128b/132b	self-synchronising	
Gigabit Ethernet (1000BASE-T)	TCM + PAM	synchronous	✓*
Gigabit Ethernet (1000BASE-X)	8b/10b	–	†
10 Gigabit Ethernet	64b/66b	self-synchronising	
PCI Express 2.0	8b/10b	synchronous	✓
PCI Express 3.0	128b/130b	self-synchronising	

Table 6.2: List of commonly used scrambled serial protocols with encoder and scrambler choices, highlighting those to which techniques described in this thesis might be adaptable. *Likely complicated by the use of five-level pulse amplitude modulation instead of two-level signalling. †Usually used over fibre, so sufficient electromagnetic leakage is unlikely.

the tenth bit. A data byte can therefore be TMDS-encoded as two (possibly identical) codewords, one used when the running disparity is positive, and the other when it is negative. Unlike 8b/10b, the running disparity cannot be treated as a single bit of state, since it can have any even value from -8 to 8.

All algorithms presented in this thesis can be adapted to TMDS-encoded scrambled data with little modification. Including only balanced symbols in templates, as in Section 5.2, allows us to demodulate signals independently of the encoder state. Out of 256 byte values, 52 are TMDS-encoded as balanced symbols that do not depend on the current disparity. Additionally, the bit indicating the choice between XOR and XNOR is not inverted and is thus independent of the disparity. On average, 28% of TMDS-encoded bits would be included in a balanced template, compared to 54% for DisplayPort. The processing gain of template correlation would, therefore, approximately be halved when compared to a 8b/10b encoding. Due to the larger number of encoder states, scrambler tracking would also become more computationally expensive, since 9 prompt correlators would be required to identify the initial state inside a blanking period.

Scrambler-based codes

Very high data rate (≥ 10 Gbit/s) digital interfaces commonly use one of several scrambler-based codes designed to have lower overhead than the 25% of 8b/10b encoding. The most common such code is 64b/66b, used in 10 Gigabit Ethernet and Thunderbolt 3. A 64-bit

symbol is first prefixed with either 01, indicating data, or 10, indicating control information possibly followed by data. The remaining 64 bits of the output symbol are produced by scrambling the input with a 58-bit self-synchronising scrambler. The encoder state is therefore 58 bits long. In some protocols, this is followed by a separate scrambler, even though 64b/66b encoded data is inherently scrambled: for example, the Ethernet data framing layer uses an additional 8-bit scrambler.

Unlike TMDS or 8b/10b encoding, 64b/66b encoding does not guarantee a bound on the output's running disparity. It instead only produces a probabilistically DC balanced output, since approximately one half of scrambled bits will be zeroes.

Other scrambler-based codes include 128b/130b, used in PCIe 3+, and 128b/132b, used in USB 3.0 and high-bitrate DisplayPort 2.0. These are based on the same approach, only with an increased payload size, and in the case of 128b/132b, a longer prefix to reduce the impact of bit errors.

Scrambler-based codes pose the same challenges to an eavesdropper as the self-synchronising scrambler they are based on. As I will discuss below, the algorithms in this thesis are not suitable for systems using self-synchronising scramblers.

6.2.2 Scrambling

Synchronous

A synchronous scrambler, such as the one used in DisplayPort, is applied to the data stream by replacing each data byte with its exclusive-or with the output of a pseudorandom sequence, usually generated by a linear-feedback shift register. The receiver maintains a replica of this LFSR, which is synchronised with the transmitter by periodically resetting the state and notifying the receiver (e.g. with a control symbol).

The algorithms in this thesis do not depend on any special properties of the DisplayPort scrambler, and would be usable with any synchronous scrambler which is reset at an (approximately) constant rate. Increasing the scrambler period by using a longer LFSR would increase the computational cost of the initial synchronisation search, but would not affect later scrambler tracking or image reconstruction.

With the 16-bit DisplayPort scrambler, this initial search is approximately 100 to 1000 times faster than the remaining image reconstruction algorithm, depending on the number of candidate colours. An at least thousandfold increase in the scrambler period would therefore be necessary to significantly increase the total computational cost. This would require the scrambler reset period to be longer than $1000 \cdot (2^{16} - 1)$ bytes, or around 3 s for a 1.62 Gbit/s interface, which would cause undesirably long startup times and data interruptions if a scrambler reset symbol is incorrectly decoded.

Self-synchronising

The state of a self-synchronising scrambler is long-term data-dependent, since the input to the LFSR is the exclusive-or of its previous output and the next data bit. The inverse operation in the receiver is exclusive-or of several previous scrambled bits, at positions determined by LFSR taps. For a degree-n LFSR, the receiver must therefore maintain a state containing the previous n scrambled bits. A one-bit change in the scrambler input will propagate through the entire remaining output stream, while a one-bit change in the output will only affect the following n bits after unscrambling.

The initial correlation search for known scrambler states in the blanking period can be adapted to a self-synchronising scrambler, since its output will cycle through a $2^n - 1$ bit periodic sequence as long as the input is constant. All later algorithms are, however, not applicable. Since the scrambler state depends on image data, successful synchronisation in the blanking period will only last until the first incorrectly identified image byte.

6.2.3 Order of operations

Scramble then encode

Almost all commonly-used protocols scramble data before encoding, either using a separate scrambler or implicitly as part of 64b/66b (or similar) encoding. The nonlinear encoder prevents us from treating the scrambler like a spreading code in a spread-spectrum signal, and so all image reconstruction algorithms in this thesis are template-based instead.

Encode then scramble

If data is first encoded and then scrambled, we can treat the two stages separately, similarly to a spread-spectrum receiver. Multiplying the received signal by a scrambler replica removes the effect of scrambling, and leaves us with a signal like that from an encoded, but not scrambled, interface. The result can then be attacked using eavesdropping techniques like those used for HDMI, such as amplitude or phase demodulation, at a much lower computational cost compared to template-based techniques. Such an attack would have the same limitation as those targetting HDMI: although computationally efficient, the colours in the resulting image do not meaningfully correspond to those displayed on the monitor.

6.3 Future work

Some possible directions for future research on DisplayPort electromagnetic eavesdropping are:

- As shown in Section 5.5, the range of eavesdropping algorithms described in this thesis is limited by the initial reset model search. A more sophisticated algorithm might include more information than just a correlation-based search for scrambler reset times. The EM algorithm that estimates the (x_0, R, P) timing parameters can also be made more robust to misidentified reset times, for example by fitting the parameters to a subset of the observed times and using the result as an initial estimate to eliminate outliers.

- My reverse-engineering of the implementation-specific padding algorithm was sufficient to develop eavesdropping algorithms and demonstrate that they can be used to target Intel interfaces. The same work can be repeated for other manufacturers, and extending it to predict lines in which fill bytes are inserted would increase the accuracy of template alignment and defeat the low-contrast colour pairs based on cyclical shifts I suggested above.

- The Julia implementation I used to produce reconstructed images in Chapter 5 is entirely CPU-based. It is parallelised and processes different colour channels on separate CPU cores, but is still two orders of magnitude slower than what would be necessary for real-time operation. This is a good candidate for a GPU-based implementation, since different colours and image lines can be processed independently.

- Another possible implementation-heavy project would be to implement the proposed software countermeasure where a custom video driver replaces the zero bytes in blanking periods with random secondary data packets.

- Ignoring leap bytes, a scrambled DisplayPort signal is periodic after $512/\gcd(512, h_\mathrm{t})$ frames, when the scrambler resets in the same lines. This period is greatest for odd h_t, and for $f_\mathrm{v} = 60$ Hz it is approximately $512/60$ Hz $= 8.53$ s. Periodic averaging with this period, before any DisplayPort-specific processing, might improve the signal-to-noise ratio and extend viable eavesdropping distance.

- The techniques in this thesis can possibly be adapted to other DisplayPort-based video interfaces: embedded DisplayPort (eDP) and DisplayPort over USB-C. My current obstacle here is the lack of public information about these protocols, since the relevant standards are only provided to members of the Video Electronics Standards Association (VESA).

- More sophisticated equipment, including higher-gain antennas and larger receivers bandwidths, could be used for longer-distance demonstrations. At receiver bandwidths significantly higher than the 50 MHz I used, choosing a reception range that is free of interference becomes difficult, and preprocessing the received IQ data to reduce narrow-band interference would likely be necessary.

Appendix A

8b/10b encoding

The 8b/10b encoder used in DisplayPort encodes each data byte or control symbol as one of two (possibly identical) ten-bit symbols, depending on the current *running disparity (RD)*. The running disparity is either +1, indicating that the output so far contained more one-bits than zero-bits, or −1, indicating that it contained more zero-bits. The symbols are chosen such that the number of zeroes and ones in the output is always either equal or differs by ±2.

Table A.1 lists all control symbols supported by the 8b/10b encoder, with their generic (ANSI) names, and for those used in DisplayPort, the names and meanings used in the DisplayPort standard. Tables A.2 and A.3 show the encodings of all 256 byte values.

Table A.1: 8b/10b encoded control symbols.

Symbol		DisplayPort meaning	Encoded	
ANSI	DP		RD = +	RD = −
K.28.0	SR	Scrambler Reset	1101 000011	0010 111100
K.28.1	CP	Copyright Protection Blanking Start	0110 000011	1001 111100
K.28.2	SS	Secondary-data Start	0101 000011	1010 111100
K.28.3	BF	Copyright Protection Scrambler Reset	0011 000011	1100 111100
K.28.5	BS	Blanking Start	1010 000011	0101 111100
K.23.7	FE	Fill End	1110 101000	0001 010111
K.27.7	BE	Blanking End	1110 100100	0001 011011
K.29.7	SE	Secondary-data End	1110 100010	0001 011101
K.30.7	FS	Fill Start	1110 100001	0001 011110
K.28.4	–		1011 000011	0100 111100
K.28.6	–		1001 000011	0110 111100
K.28.7	–		1110 000011	0001 111100

Table A.2: 8b/10b encoded byte values from `0x00` to `0x7f`.

	Bits	Encoded			Bits	Encoded			Bits	Encoded			Bits	Encoded	
		RD = +	RD = −			RD = +	RD = −			RD = +	RD = −			RD = +	RD = −
00	000 00000	1101 000110	0010 111001	20	001 00000	1001 000110	1001 111001	40	010 00000	1010 000110	1010 111001	60	011 00000	0011 000110	1100 111001
01	000 00001	1101 010001	0010 101110	21	001 00001	1001 010001	1001 101110	41	010 00001	1010 010001	1010 101110	61	011 00001	0011 010001	1100 101110
02	000 00010	1101 010010	0010 101101	22	001 00010	1001 010010	1001 101101	42	010 00010	1010 010010	1010 101101	62	011 00010	0011 010010	1100 101101
03	000 00011	0010 100011	1101 100011	23	001 00011	1001 100011		43	010 00011	1010 100011		63	011 00011	1100 100011	0011 100011
04	000 00100	1101 010100	0010 101011	24	001 00100	1001 010100	1001 101011	44	010 00100	1010 010100	1010 101011	64	011 00100	0011 010100	1100 101011
05	000 00101	0010 100101	1101 100101	25	001 00101	1001 100101		45	010 00101	1010 100101		65	011 00101	1100 100101	0011 100101
06	000 00110	0010 100110	1101 100110	26	001 00110	1001 100110		46	010 00110	1010 100110		66	011 00110	1100 100110	0011 100110
07	000 00111	0010 111000	1101 000111	27	001 00111	1001 111000	1001 000111	47	010 00111	1010 111000	1010 000111	67	011 00111	1100 111000	0011 000111
08	000 01000	1101 011000	0010 100111	28	001 01000	1001 011000	1001 100111	48	010 01000	1010 011000	1010 100111	68	011 01000	0011 011000	1100 100111
09	000 01001	0010 101001	1101 101001	29	001 01001	1001 101001		49	010 01001	1010 101001		69	011 01001	1100 101001	0011 101001
0a	000 01010	0010 101010	1101 101010	2a	001 01010	1001 101010		4a	010 01010	1010 101010		6a	011 01010	1100 101010	0011 101010
0b	000 01011	0010 001011	1101 001011	2b	001 01011	1001 001011		4b	010 01011	1010 001011		6b	011 01011	1100 001011	0011 001011
0c	000 01100	0010 101100	1101 101100	2c	001 01100	1001 101100		4c	010 01100	1010 101100		6c	011 01100	1100 101100	0011 101100
0d	000 01101	0010 001101	1101 001101	2d	001 01101	1001 001101		4d	010 01101	1010 001101		6d	011 01101	1100 001101	0011 001101
0e	000 01110	0010 001110	1101 001110	2e	001 01110	1001 001110		4e	010 01110	1010 001110		6e	011 01110	1100 001110	0011 001110
0f	000 01111	1101 000101	0010 111010	2f	001 01111	1001 000101	1001 111010	4f	010 01111	1010 000101	1010 111010	6f	011 01111	0011 000101	1100 111010
10	000 10000	1101 001001	0010 110110	30	001 10000	1001 001001	1001 110110	50	010 10000	1010 001001	1010 110110	70	011 10000	0011 001001	1100 110110
11	000 10001	0010 110001	1101 110001	31	001 10001	1001 110001		51	010 10001	1010 110001		71	011 10001	1100 110001	0011 110001
12	000 10010	0010 110010	1101 110010	32	001 10010	1001 110010		52	010 10010	1010 110010		72	011 10010	1100 110010	0011 110010
13	000 10011	0010 010011	1101 010011	33	001 10011	1001 010011		53	010 10011	1010 010011		73	011 10011	1100 010011	0011 010011
14	000 10100	0010 110100	1101 110100	34	001 10100	1001 110100		54	010 10100	1010 110100		74	011 10100	1100 110100	0011 110100
15	000 10101	0010 010101	1101 010101	35	001 10101	1001 010101		55	010 10101	1010 010101		75	011 10101	1100 010101	0011 010101
16	000 10110	0010 010110	1101 010110	36	001 10110	1001 010110		56	010 10110	1010 010110		76	011 10110	1100 010110	0011 010110
17	000 10111	1101 101000	0010 010111	37	001 10111	1001 101000	1001 010111	57	010 10111	1010 101000	1010 010111	77	011 10111	0011 101000	1100 010111
18	000 11000	1101 001100	0010 110011	38	001 11000	1001 001100	1001 110011	58	010 11000	1010 001100	1010 110011	78	011 11000	0011 001100	1100 110011
19	000 11001	0010 011001	1101 011001	39	001 11001	1001 011001		59	010 11001	1010 011001		79	011 11001	1100 011001	0011 011001
1a	000 11010	0010 011010	1101 011010	3a	001 11010	1001 011010		5a	010 11010	1010 011010		7a	011 11010	1100 011010	0011 011010
1b	000 11011	1101 100100	0010 011011	3b	001 11011	1001 100100	1001 011011	5b	010 11011	1010 100100	1010 011011	7b	011 11011	0011 100100	1100 011011
1c	000 11100	0010 011100	1101 011100	3c	001 11100	1001 011100		5c	010 11100	1010 011100		7c	011 11100	1100 011100	0011 011100
1d	000 11101	1101 100010	0010 011101	3d	001 11101	1001 100010	1001 011101	5d	010 11101	1010 100010	1010 011101	7d	011 11101	0011 100010	1100 011101
1e	000 11110	1101 100001	0010 011110	3e	001 11110	1001 100001	1001 011110	5e	010 11110	1010 100001	1010 011110	7e	011 11110	0011 100001	1100 011110
1f	000 11111	1101 001010	0010 110101	3f	001 11111	1001 001010	1001 110101	5f	010 11111	1010 001010	1010 110101	7f	011 11111	0011 001010	1100 110101

Table A.3: 8b/10b encoded byte values from `0x80` to `0xff`.

	Bits	Encoded			Bits	Encoded			Bits	Encoded			Bits	Encoded	
		RD = +	RD = −			RD = +	RD = −			RD = +	RD = −			RD = +	RD = −
80	100 00000	1011 000110	0100 111001	a0	101 00000	0101 000110	0101 111001	c0	110 00000	0110 000110	0110 111001	e0	111 00000	0111 000110	1000 111001
81	100 00001	1011 010001	0100 101110	a1	101 00001	0101 010001	0101 101110	c1	110 00001	0110 010001	0110 101110	e1	111 00001	0111 010001	1000 101110
82	100 00010	1011 010010	0100 101101	a2	101 00010	0101 010010	0101 101101	c2	110 00010	0110 010010	0110 101101	e2	111 00010	0111 010010	1000 101101
83	100 00011	0100 100011	1011 100011	a3	101 00011	0101 100011		c3	110 00011	0110 100011		e3	111 00011	1000 100011	0111 100011
84	100 00100	1011 010100	0100 101011	a4	101 00100	0101 010100	0101 101011	c4	110 00100	0110 010100	0110 101011	e4	111 00100	0111 010100	1000 101011
85	100 00101	0100 100101	1011 100101	a5	101 00101	0101 100101		c5	110 00101	0110 100101		e5	111 00101	1000 100101	0111 100101
86	100 00110	0100 100110	1011 100110	a6	101 00110	0101 100110		c6	110 00110	0110 100110		e6	111 00110	1000 100110	0111 100110
87	100 00111	0100 111000	1011 000111	a7	101 00111	0101 111000	0101 000111	c7	110 00111	0110 111000	0110 000111	e7	111 00111	1000 111000	0111 000111
88	100 01000	1011 011000	0100 100111	a8	101 01000	0101 011000	0101 100111	c8	110 01000	0110 011000	0110 100111	e8	111 01000	0111 011000	1000 100111
89	100 01001	0100 101001	1011 101001	a9	101 01001	0101 101001		c9	110 01001	0110 101001		e9	111 01001	1000 101001	0111 101001
8a	100 01010	0100 101010	1011 101010	aa	101 01010	0101 101010		ca	110 01010	0110 101010		ea	111 01010	1000 101010	0111 101010
8b	100 01011	0100 001011	1011 001011	ab	101 01011	0101 001011		cb	110 01011	0110 001011		eb	111 01011	0001 001011	0111 001011
8c	100 01100	0100 101100	1011 101100	ac	101 01100	0101 101100		cc	110 01100	0110 101100		ec	111 01100	1000 101100	0111 101100
8d	100 01101	0100 001101	1011 001101	ad	101 01101	0101 001101		cd	110 01101	0110 001101		ed	111 01101	0001 001101	0111 001101
8e	100 01110	0100 001110	1011 001110	ae	101 01110	0101 001110		ce	110 01110	0110 001110		ee	111 01110	0001 001110	0111 001110
8f	100 01111	1011 000101	0100 111010	af	101 01111	0101 000101	0101 111010	cf	110 01111	0110 000101	0110 111010	ef	111 01111	0111 000101	1000 111010
90	100 10000	1011 001001	0100 110110	b0	101 10000	0101 001001	0101 110110	d0	110 10000	0110 001001	0110 110110	f0	111 10000	0111 001001	1000 110110
91	100 10001	0100 110001	1011 110001	b1	101 10001	0101 110001		d1	110 10001	0110 110001		f1	111 10001	1000 110001	1110 110001
92	100 10010	0100 110010	1011 110010	b2	101 10010	0101 110010		d2	110 10010	0110 110010		f2	111 10010	1000 110010	1110 110010
93	100 10011	0100 010011	1011 010011	b3	101 10011	0101 010011		d3	110 10011	0110 010011		f3	111 10011	1000 010011	0111 010011
94	100 10100	0100 110100	1011 110100	b4	101 10100	0101 110100		d4	110 10100	0110 110100		f4	111 10100	1000 110100	1110 110100
95	100 10101	0100 010101	1011 010101	b5	101 10101	0101 010101		d5	110 10101	0110 010101		f5	111 10101	1000 010101	0111 010101
96	100 10110	0100 010110	1011 010110	b6	101 10110	0101 010110		d6	110 10110	0110 010110		f6	111 10110	1000 010110	0111 010110
97	100 10111	1011 101000	0100 010111	b7	101 10111	0101 101000	0101 010111	d7	110 10111	0110 101000	0110 010111	f7	111 10111	0111 101000	1000 010111
98	100 11000	1011 001100	0100 110011	b8	101 11000	0101 001100	0101 110011	d8	110 11000	0110 001100	0110 110011	f8	111 11000	0111 001100	1000 110011
99	100 11001	0100 011001	1011 011001	b9	101 11001	0101 011001		d9	110 11001	0110 011001		f9	111 11001	1000 011001	0111 011001
9a	100 11010	0100 011010	1011 011010	ba	101 11010	0101 011010		da	110 11010	0110 011010		fa	111 11010	1000 011010	0111 011010
9b	100 11011	1011 100100	0100 011011	bb	101 11011	0101 100100	0101 011011	db	110 11011	0110 100100	0110 011011	fb	111 11011	0111 100100	1000 011011
9c	100 11100	0100 011100	1011 011100	bc	101 11100	0101 011100		dc	110 11100	0110 011100		fc	111 11100	1000 011100	0111 011100
9d	100 11101	1011 100010	0100 011101	bd	101 11101	0101 100010	0101 011101	dd	110 11101	0110 100010	0110 011101	fd	111 11101	0111 100010	1000 011101
9e	100 11110	1011 100001	0100 011110	be	101 11110	0101 100001	0101 011110	de	110 11110	0110 100001	0110 011110	fe	111 11110	0111 100001	1000 011110
9f	100 11111	1011 001010	0100 110101	bf	101 11111	0101 001010	0101 110101	df	110 11111	0110 001010	0110 110101	ff	111 11111	0111 001010	1000 110101

Appendix B

Farrow filter

The Farrow filter [91] is a polyphase structure that allows us to preprocess a discrete-time sequence $x[n]$, and then efficiently sample the corresponding continuous, sinc-interpolated signal $s(t)$ at arbitrary real-valued offsets t. It is based on a polynomial approximation of coefficients of a fractional delay filter, which shift their discrete-time input by $0 \leq \delta < 1$ (left half of Figure B.1). Convolving $x[n]$ with the array of polynomial coefficients for each term gives a polynomial approximation for $s(t)$ (bottom right).

Let f_s be the sampling rate of $x[n]$. The interpolated signal can be computed by convolving with a scaled sinc function

$$s(t) = \sum_i x[i] \operatorname{sinc}\left(\frac{t - i f_\mathrm{s}^{-1}}{f_\mathrm{s}^{-1}}\right) \tag{B.1}$$

where $\operatorname{sinc} x = \frac{\sin \pi x}{\pi x}$.

If we write $t = (n + \delta) f_\mathrm{s}^{-1}$ as a sum of an integer part n and a fractional part $0 \leq \delta < 1$ (in samples), this is equivalent to taking the n-th element of the discrete-time convolution of $x[n]$ and an impulse response h_δ with coefficients

$$h_\delta[n] = \operatorname{sinc}\left(\frac{\delta - n f_\mathrm{s}^{-1}}{f_\mathrm{s}^{-1}}\right) = \operatorname{sinc}\left(\delta f_\mathrm{s} - n\right) \tag{B.2}$$

which shifts the signal by a fractional delay δ.

This impulse response is infinitely long for $\delta \neq 0$, but values for large n tend to zero, since $|\operatorname{sinc}(n)| < n$ for $n \neq 0$. Using the usual approach for designing a windowed-sinc FIR filter, we can truncate the impulse response by keeping coefficients $h[n]$ where $\frac{M}{2} \leq n \leq \frac{M}{2}$, for a desired filter order M, and set all other coefficients to zero.

Now consider a collection of R such fractional delay filters $h_{0/R}[n], h_{1/R}[n], \ldots, h_{(R-1)/R}[n]$, where the m-th filter shifts its input by $\frac{m}{R} f_\mathrm{s}$. Using such a collection, we could sample $s(t)$ at any $t = (n + \frac{m}{R}) f_\mathrm{s}^{-1}$ for $n, m \in \mathbb{Z}$ by taking the n-th sample of the convolution of x and h_m.

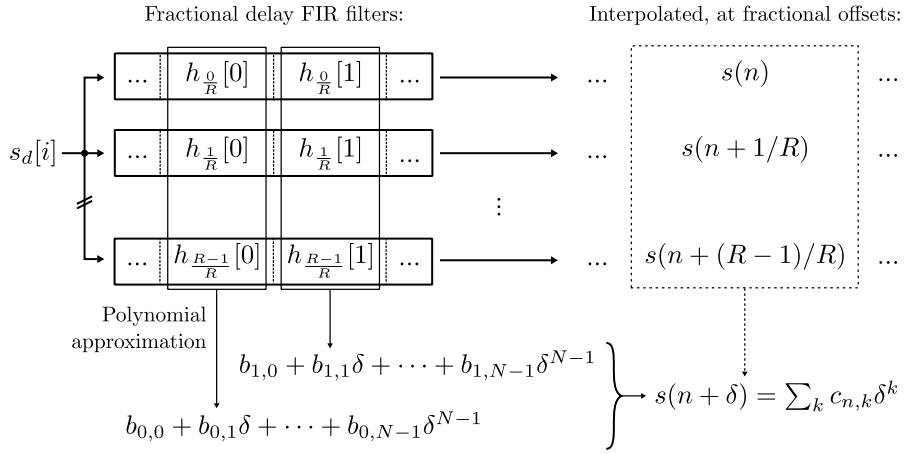


Figure B.1: Construction of a Farrow filter from R fractional delay filters, using degree $N-1$ interpolating polynomials. For simplicity, the sampling rate is omitted (assuming $f_\mathrm{s}=1$).

The Farrow filter allows us to sample at other offsets by interpolating between the fractional delay filters using a degree $N-1$ polynomial (i.e. with N coefficients). Consider each index n separately, and compute an approximation to $h_\delta[n]$, the n-th impulse response coefficient of a filter that delays the input by δf_s^{-1}, by interpolating the known values $h_{m/R}[n]$ to obtain a series of coefficients $b_{n,k}$ where

$$h_\delta[n] \approx b_{n,0} + b_{n,1}\delta + b_{n,2}\delta^2 + \cdots + b_{n,N-1}\delta^{N-1}. \tag{B.3}$$

We can now compute $s(t)$ by convolving with $h_\delta[n]$:

$$s(t) = s((n+\delta)f_\mathrm{s}^{-1}) = \sum_{i=-M/2}^{M/2} x[n-i]h_\delta[i] = \sum_{i=-M/2}^{M/2} x[n-i]\sum_{k=0}^{N-1} b_{i,k}\delta^k \tag{B.4}$$

This is still inefficient, requiring $\mathcal{O}(NM)$ multiplications for a single evaluation of $s(t)$. We can simplify the expression by swapping the order of the sums

$$s(t) = \sum_{k=0}^{N-1}\sum_{i=-M/2}^{M/2} x[n-i]b_{i,k}\delta^k \tag{B.5}$$

and introducing a new set of coefficients $c_{n,k}$ defined as

$$c_{n,k} = \sum_{i=-M/2}^{M/2} x[n-i]b_{i,k} \tag{B.6}$$

to obtain a simplified expression

$$s(t) = \sum_{k=0}^{N-1} c_{n,k}\delta^k \tag{B.7}$$

that can now be evaluated with only N additions and multiplications.

To sum up, we preprocess $x[n]$ by creating R order-M fractional delay filters, interpolating their coefficients with degree-$N-1$ polynomials, and computing the sums $c_{n,k}$ for all indices n in $x[n]$ and $0 \leq k < N$. These sums can be efficiently computed with k convolutions of $x[n]$ and a length-$M+1$ filter kernel, in $\mathcal{O}(N|x| \log |x|)$ time. Once the coefficients are available, $s(t)$ can be easily computed in $\mathcal{O}(N)$.

Note that since the fractional delay filter order M only affects the preprocessing time, which is dominated by a convolution with $x[n]$ (where $|x| \gg M$), we can use a long impulse response to minimise distortion added by truncating. In my implementation, I use $M = 351$. Similarly, the number of filters R only affects preprocessing time, and I set $R = 20M$. The cost of evaluation $s(t)$ only depends on the interpolation degree N, where I use $N = 5$.

Bibliography

[1] Markus G. Kuhn. Tempest. In Sushil Jajodia, Pierangela Samarati, and Moti Yung, editors, *Encyclopedia of Cryptography, Security and Privacy*. Springer, Berlin, Heidelberg, 2023.

[2] Wim van Eck. Electromagnetic radiation from video display units: an eavesdropping risk? *Computers & Security*, 4(4):269–286, 1985.

[3] Markus G. Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In David Martin and Andrei Serjantov, editors, *Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pages 88–107, Berlin, Heidelberg, 2005. Springer.

[4] Ankur Verma. AN-1032: An introduction to FPD-Link. Texas Instruments, 1998.

[5] Open LVDS Display Interface (OpenLDI) specification. National Semiconductor, 1999.

[6] Digital Visual Interface DVI, revision 1.0. Digital Display Working Group, 1999.

[7] High-Definition Multimedia Interface, specification version 1.3a. HDMI Licensing, LLC, 2006.

[8] VESA DisplayPort standard, version 1, revision 2. Video Electronics Standards Association, January 2010.

[9] Ireneusz Kubiak and Artur Przybysz. DVI (HDMI) and DisplayPort digital video interfaces in electromagnetic eavesdropping process. In *2019 International Symposium on Electromagnetic Compatibility – EMC EUROPE*, pages 388–393, September 2019. ISSN: 2325-0364.

[10] Markus G. Kuhn. Ballot printer – protection against eavesdropping attacks – guidance for system designers. Bijlage 2016D17984: Richtlijnen ter voorkoming van compromitterende straling. Tweede Kamer der Staten-Generaal (House of Representatives, Netherlands), kamerstuk, Apr 2016.

[11] Major-General RFH Nalder. *The Royal Corps of Signals*. Royal Signals Institution, 1958.

[12] David G. Boak. *A History Of U.S. Communications Security.* National Security Agency, July 1973. Declassified 2008-10-12.

[13] David Easter. The impact of 'Tempest' on Anglo-American communications security and intelligence, 1943–1970. *Intelligence and National Security*, 36(1):1–16, 2021.

[14] National Security Agency. Engstrom study for proposed Office of Research in AFSA; task list for Office of Research and Development. William F. Friedman Collection of Official Papers (Folder 369), 1952–1954. Declassified 2015-04-20.

[15] Peter Wright and Paul Greengrass. *The candid autobiography of a senior intelligence officer.* Heinemann, 1987.

[16] TEMPEST timeline. `https://cryptome.org/tempest-time.htm`, 2002. Accessed: 2023-09-14.

[17] Ross Anderson. *Security engineering: a guide to building dependable distributed systems.* John Wiley & Sons, 2020.

[18] Willis H. Ware. *Security and Privacy in Computer Systems.* RAND Corporation, Santa Monica, CA, 1967.

[19] Han Fang. Radiated emission from CRT of computer VDU. In *IEEE International Symposium on Electromagnetic Compatibility*, pages 58–61, August 1990.

[20] N. E. Koksaldi, I. Olcer, U. Yapanel, and U. Sarac. Signal processing applications for information extraction from the radiation of VDUs. October 1998.

[21] N. E. Koksaldi, S. S. Seker, and B. Sankur. Information extraction from the radiation of VDUs by pattern recognition methods. 2000.

[22] Peter Smulders. The threat of information theft by reception of electromagnetic radiation from RS-232 cables. *Computers & Security*, 9(1):53–58, February 1990.

[23] Hidema Tanaka, Osamu Takizawa, and Akihiro Yamamura. A trial of the interception of display image using emanation of electromagnetic wave. *Journal of the National Institute of Information and Communications Technology*, 52(1/2):213–223, 2005.

[24] Fürkan Elibol, Uğur Sarac, and Işin Erer. Realistic eavesdropping attacks on computer displays with low-cost and mobile receiver system. In *Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, pages 1767–1771, 2012.

[25] Martin Marinov. Remote video eavesdropping using a software-defined radio platform. Master's thesis, University of Cambridge, Computer Laboratory, June 2014.

[26] Christian David O'Connell. *Exploiting quasiperiodic electromagnetic radiation using software-defined radio*. PhD thesis, University of Cambridge, Computer Laboratory, 2019.

[27] Ho Seong Lee, Dong Hoon Choi, Kyuhong Sim, and Jong-Gwan Yook. Information recovery using electromagnetic emanations from display devices under realistic environment. *IEEE Transactions on Electromagnetic Compatibility*, 61(4):1098–1106, August 2019. Conference Name: IEEE Transactions on Electromagnetic Compatibility.

[28] Pieterjan de Meulemeester, Bart Scheers, and Guy A.E. Vandenbosch. Eavesdropping a (ultra-)high-definition video display from an 80 meter distance under realistic circumstances. In *2020 IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity (EMCSI)*, pages 517–522, July 2020.

[29] Pieterjan De Meulemeester, Bart Scheers, and Guy A.E. Vandenbosch. Differential signaling compromises video information security through AM and FM leakage emissions. *IEEE Transactions on Electromagnetic Compatibility*, 62(6):2376–2385, December 2020.

[30] Pieterjan De Meulemeester, Bart Scheers, and Guy A.E. Vandenbosch. Reconstructing video images in color exploiting compromising video emanations. In *2020 International Symposium on Electromagnetic Compatibility – EMC EUROPE*, pages 1–6, September 2020. ISSN: 2325-0364.

[31] Dong-Hoon Choi, Euibum Lee, and Jong-Gwan Yook. Reconstruction of video information through leakaged electromagnetic waves from two VDUs using a narrow band-pass filter. *IEEE Access*, 10:40307–40315, 2022.

[32] Ireneusz Kubiak and Artur Przybysz. Fourier and chirp-Z transforms in the estimation values process of horizontal and vertical synchronization frequencies of graphic displays. *Applied Sciences*, 12(10):5281, January 2022. Number: 10 Publisher: Multidisciplinary Digital Publishing Institute.

[33] Florian Lemarchand, Cyril Marlin, Florent Montreuil, Erwan Nogues, and Maxime Pelcat. Electro-magnetic side-channel attack through learned denoising and classification. In *ICASSP 2020 – 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2882–2886, May 2020. ISSN: 2379-190X.

[34] J. Galvis, S. Morales, C. Kasmi, and F. Vega. Denoising of video frames resulting from video interface leakage using deep learning for efficient optical character recognition. *IEEE Letters on Electromagnetic Compatibility Practice and Applications*, 3(2):82–86, June 2021. Conference Name: IEEE Letters on Electromagnetic Compatibility Practice and Applications.

[35] Zhengxiong Li, Fenglong Ma, Aditya Singh Rathore, Zhuolin Yang, Baicheng Chen, Lu Su, and Wenyao Xu. WaveSpy: Remote and through-wall screen attack via mmWave sensing. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 217–232, May 2020. ISSN: 2375-1207.

[36] Wenqiang Jin, Srinivasan Murali, Huadi Zhu, and Ming Li. Periscope: a keystroke inference attack using human coupled electromagnetic emanations. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 700–714. ACM, November 2021.

[37] Markus G. Kuhn. Compromising emanations of LCD TV sets. *IEEE Transactions on Electromagnetic Compatibility*, 55(3):564–570, June 2013.

[38] Yuichi Hayashi, Naofumi Homma, Mamoru Miura, Takafumi Aoki, and Hideaki Sone. A threat for tablet PCs in public space: Remote visualization of screen images using EM emanation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 954–965, New York, NY, USA, November 2014. Association for Computing Machinery.

[39] Ireneusz Kubiak and Joe Loughry. LED arrays of laser printers as valuable sources of electromagnetic waves for acquisition of graphic data. *Electronics*, 8(10):1078, 2019.

[40] Alireza Nazari, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. EDDIE: EM-based detection of deviations in program execution. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 333–346, New York, NY, USA, June 2017. Association for Computing Machinery.

[41] Ziwei Liu, Feng Lin, Chao Wang, Yijie Shen, Zhongjie Ba, Li Lu, Wenyao Xu, and Kui Ren. CamRadar: Hidden camera detection leveraging amplitude-modulated sensor images embedded in electromagnetic emanations. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(4):173:1–173:25, January 2023.

[42] Yihua Peng, Jiemin Zhang, Jian Mao, and Mengmeng Cui. A signal-denoising method for electromagnetic leakage from USB keyboards. *Electronics*, 12(17):3647, 2023.

[43] Yasunao Suzuki and Yoshiharu Akiyama. Jamming technique to prevent information leakage caused by unintentional emissions of PC video signals. In *2010 IEEE International Symposium on Electromagnetic Compatibility*, pages 132–137, July 2010. ISSN: 2158-1118.

[44] Tae-Lim Song, Yi-Ru Jeong, Han-Shin Jo, and Jong-Gwan Yook. Noise-jamming effect as a countermeasure against TEMPEST during high-speed signaling. *IEEE*

Transactions on Electromagnetic Compatibility, 57(6):1491–1500, December 2015. Conference Name: IEEE Transactions on Electromagnetic Compatibility.

[45] Markus G. Kuhn and Ross J. Anderson. Soft Tempest: Hidden data transmission using electromagnetic emanations. In David Aucsmith, editor, *Information Hiding*, Lecture Notes in Computer Science, pages 124–142, Berlin, Heidelberg, 1998. Springer.

[46] Hidema Tanaka. Information leakage via electromagnetic emanations and evaluation of Tempest countermeasures. In Patrick McDaniel and Shyam K. Gupta, editors, *Information Systems Security*, Lecture Notes in Computer Science, pages 167–179, Berlin, Heidelberg, 2007. Springer.

[47] Ireneusz Kubiak. *Computer font resistant to electromagnetic infiltration.* Publisher House of Military University of Technology, Warsaw, January 2014.

[48] Bogdan Trip, Vlad Butnariu, Mădălin Vizitiu, Alexandru Boitan, and Simona Halunga. Analysis of compromising video disturbances through power line. *Sensors*, 22(1):267, January 2022. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[49] Stanislav Subbotin, Dmitry Volchkov, Alexander Zabokritski, and Elena Dymova. Justification of the relevance of developing a test program for special studies of the DisplayPort interface. In *2021 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, pages 0500–0502, May 2021.

[50] Egor A. Simakhin, Danil A. Shinyaev, Igor I. Kagin, Leonid N. Kessarinskiy, and Anatoly P. Durakovskiy. Analysis of electromagnetic radiation of LCD monitor with DisplayPort interface. In *2022 Moscow Workshop on Electronic and Networking Technologies (MWENT)*, pages 1–5, June 2022.

[51] VESA Coordinated Video Timings (CVT) standard, version 1.2. Video Electronics Standards Association, February 2013.

[52] VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT) standard, version 1, rev. 12. Video Electronics Standards Association, November 2008.

[53] Daniel Fleisch. *A student's guide to Maxwell's equations.* Cambridge University Press, 2008.

[54] Tze-Chuen Toh. *Electromagnetic theory for electromagnetic compatibility engineers.* CRC Press, 2013.

[55] Clayton R Paul. *Introduction to electromagnetic compatibility.* John Wiley & Sons, 2006.

[56] Markus G. Kuhn. Compromising emanations: eavesdropping risks of computer displays. Technical Report UCAM-CL-TR-577, University of Cambridge, Computer Laboratory, December 2003.

[57] Recommendation ITU-R P.372-7: Radio noise. Technical report, International Telecommunication Union, 2001.

[58] Recommendation ITU-R P.372-16: Radio noise. Technical report, International Telecommunication Union, 2022.

[59] Directive 2014/30/EU of the European Parliament and of the Council of 26 February 2014 on the harmonisation of the laws of the Member States relating to electromagnetic compatibility (recast). *OJ*, L 96:79–106, 2014-03-29.

[60] Code of Federal Regulations, Part 15 – Radio frequency devices, 1989-04-25.

[61] CISPR 22:2008 Information technology equipment – radio disturbance characteristics – limits and methods of measurement. International Electrotechnical Commission, 2008-09-24.

[62] CISPR 32:2015 Electromagnetic compatibility of multimedia equipment – emission requirements. International Electrotechnical Commission, 2015-03-31.

[63] T Frenzel, J Rohde, and J Opfer. Elektromagnetische Schirmung von Gebäuden – Theoretische Grundlagen. Technical Report BSI TR-03209-1, Bundesamt für Sicherheit in der Informationstechnik, 2008.

[64] T Frenzel, J Rohde, and J Opfer. Elektromagnetische Schirmung von Gebäuden – Praktische Messungen. Technical Report BSI TR-03209-2, Bundesamt für Sicherheit in der Informationstechnik, 2008.

[65] R&S FSWT test receiver: TEMPEST measuring receiver with digital signal evaluation. Rohde & Schwarz, 2014. `https://www.rohde-schwarz.com/us/brochure-datasheet/fswt/`.

[66] Ettus Technologies. USRP hardware driver and USRP manual – USRP X3x0 series. `https://files.ettus.com/manual/page_usrp_x3x0.html`. Accessed: 2023-09-18.

[67] Ettus Technologies. USRP hardware driver and USRP manual – daughterboards. `https://files.ettus.com/manual/page_dboards.html`. Accessed: 2023-09-18.

[68] Karl Rothammel and Alois Krischke. *Rothammels Antennenbuch*. Franckh-Kosmos, 1995.

[69] F Timischl. The contrast-to-noise ratio for image quality evaluation in scanning electron microscopy. *Scanning*, 37(1):54–62, 2015.

[70] Marijke Welvaert and Yves Rosseel. On the definition of signal-to-noise ratio and contrast-to-noise ratio for fMRI data. *PloS one*, 8(11):e77089, 2013.

[71] Jack E Bresenham. Algorithm for computer control of a digital plotter. In *Seminal graphics: pioneering efforts that shaped the field*, pages 1–6. 1998.

[72] IEEE standard for Ethernet. *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, pages 1404–1679, 2018.

[73] Pratap Misra and Per Enge. *Global Positioning System: Signal, measurement, and performance*. Ganga-Jamuna Press, 2004, 2004.

[74] Elliott D. Kaplan and Christopher J. Hegarty. *Understanding GPS/GNSS: Principles and Applications*. Artech House, 2017.

[75] Ettus Technologies. USRP X300 and X310 series spec sheet. `https://www.ettus.com/wp-content/uploads/2019/01/X300_X310_Spec_Sheet_9.20.2022.pdf`. Accessed: 2023-09-27.

[76] Robert Feldt and Alexey Stukalov. BlackBoxOptim.jl, v0.6.2. `https://github.com/robertfeldt/BlackBoxOptim.jl`, 2023.

[77] G Jovanovic Dolecek and Sanjit K Mitra. Simple method for compensation of CIC decimation filter. *Electronics Letters*, 44(19):1, 2008.

[78] Wikipedia. Tempest (codename) – Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Tempest%20(codename)&oldid=1175260499`, 2023. Accessed: 2023-05-26.

[79] James L Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, page 204. IEEE, 1994.

[80] Boris Köpf and David Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 286–296, 2007.

[81] Kodak Lossless True Color Image Suite. Eastman Kodak Company. Photos by Steve Kelly and Alfons Rudolph. `https://r0k.us/graphics/kodak/`

[82] Interelectronix. TEMPEST. `https://www.interelectronix.com/tempest.html`. Accessed: 2023-10-11.

[83] National Security Telecommunications and Information Systems Security Committee. NSTISSAM TEMPEST/1-92. Technical report, National Security Agency, 1922. Declassified 1999-10-21.

[84] NACSIM 5000 – TEMPEST fundamentals. Technical report, National Security Agency. Declassified 2000-12-18.

[85] CTA-861-G: A DTV profile for uncompressed high speed digital interfaces. Consumer Technology Association, November 2016.

[86] Scott Crosby, Ian Goldberg, Robert Johnson, Dawn Song, and David Wagner. A cryptanalysis of the high-bandwidth digital content protection system. In *Security and Privacy in Digital Rights Management: ACM CCS-8 Workshop DRM 2001 Philadelphia, PA, USA, November 5, 2001 Revised Papers*, pages 192–200. Springer, 2002.

[87] Rob Johnson, Mikhail Rubnich, and Andres DelaCruz. Implementing a key recovery attack on the high-bandwidth digital content protection protocol. In *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 313–317. IEEE, 2011.

[88] Ars Technica. Intel confirms HDCP key is real, can now be broken at will. `https://arstechnica.com/tech-policy/2010/09/intel-confirms-the-hdcp-key-is-real-can-now-be-broken-at-will/`. Accessed: 2023-10-11.

[89] High-bandwidth Digital Content Protection System – Mapping HDCP to DisplayPort, revision 2.3. Digital Content Protection LLC, January 2019.

[90] HD Fury. `https://www.hdfury.com/`. Accessed: 2023-10-11.

[91] C.W. Farrow. A continuously variable digital delay element. In *1988., IEEE International Symposium on Circuits and Systems*, pages 2641–2645 vol.3, 1988.