Remote video eavesdropping using a software-defined radio platform

Martin Marinov
St Edmund's College

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: mtm46@cam.ac.uk

June 7, 2014

Declaration

I Martin Marinov of St Edmund's College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count:

`12,004`

Signed:

Date:

Abstract

This dissertation presents a software toolkit for remotely eavesdropping video monitors using a Software Defined Radio (SDR) receiver. It exploits compromising emanations from cables carrying video signals. Analogue video is usually transmitted one line of pixels at a time encoded as a varying current. This generates a wideband electromagnetic wave that can be picked up by an SDR receiver. The presented software can map the received field strength of each pixel to a grayscale value in order to show a real-time false colour estimate of the original video signal.

The software significantly lowers the costs required for undertaking a practical attack compared to existing solutions. Furthermore, it allows for an additional digital post-processing which can aid in analysing and improving the results. It also provides mobility for a potential adversary, requiring only a commodity laptop and an USB SDR dongle. The attacker does not need to have any prior knowledge about the victim's video display. All parameters such as resolution and refresh rate can be estimated with the aid of the software.

The software comprises of a library written in C, a collection of plug-ins for various Software Define Radio (SDR) frontends and a Java based Graphical User Interface (GUI). It is designed to be a multi-platform application. All native libraries can be pre-compiled and packed into a single Java jar file which allows the toolkit to run on any supported operating system.

This report documents the digital processing techniques that have been employed in order to extract, detect and lock to a video signal. It also explains the architecture of the software system and the techniques used in order to achieve low latency and real-time interactivity. It demonstrates the usage of the system by performing a practical attack. It then gives some ideas about what could be improved further and some analysis of data that was collected during the development of the software.

Contents

Chapter 1

Introduction

Interference is usually regarded as a simple annoyance. It is not difficult to experience TV signal or radio station getting distorted because of a device operating in the vicinity of your receiver. Manufacturers have to meet certain requirements so that their products do not cause interference with other systems. The truth is that the emanation is linked to the way the emitting device works internally. The interference is generated by alternating currents due to switches, oscillators and other electronic and mechanical components operating inside the device. Therefore it carries some information about the internal state of the device.

This means that a simple annoyance could potentially turn into a security leak, broadcasting sensitive data into the wild. A clever attacker could possibly distinguish different modes of operation or even recover raw data that is being processed. And the industry standards that manufacturers follow to minimise interference say nothing about the information such interference may carry. A very sensitive detector can pick up such signals at levels several magnitudes lower than what industry standards specify as radiation limits.

In the case of video displays the issue is particularly important. Having a secure, encrypted computer system that unintentionally broadcasts its display in clear text does not sound secure at all. The attacker could be fully

passive and the breach could be never detected. Furthermore, the repeating nature of the signal combined with the long video wires that could act as antennas, mean that such signals can travel very long distances, allowing practical attacks from vans across the street [1].

However very little research has been published into the open literature on the topic. The main reason is possibly the expensive specialised equipment required to conduct experiments with compromising emanations. One of the goals of this project is to address this issue and demonstrate that a practical attack could be undertaken using an affordable software-defined radio receiver.

1.1 History

A compass needle points north when put next to a wire that has current flowing in it. Turning off the current makes the needle twitch i.e. temporary deflecting from the north direction. The same effect could be seen if the current is turned back on. This phenomenon was noticed by the Danish physicist Hans Christian Ørsted in April 1820 [2]. It shows that a changing electric current creates a magnetic field and vice versa. Later this phenomenon was utilised to create the electrical telegraph and allow for long distance communications.

However, the undesired effects of this phenomenon received almost no attention for a couple of decades. British army noticed crosstalk between telephone wires during the Nile and Suakin expedition in 1884-85 [3]. The first reported exploitation of the phenomenon was in 1914 when earth leakage from telephone wires caused a lot of crosstalk. Listening posts were established in order to intercept enemy messages. Next year valve amplifiers were utilised which allowed for extended listening range [4].

The US National Security Agency conducted a classified research with codename TEMPEST in 1972 [5]. The document was later partially declassified in 2007. It describes unwanted emanations coming from a Bell-telephone

mixing device that was used for encryption. The signal allowed an attacker to reconstruct the original plain text. Researchers at Bell Lab demonstrated a practical attack from about 80 feet away that allowed reconstructing about 75% of the plain text being processed by the machine.

Computer monitors started to become common in the end of the 20th century. Wim van Eck published the first unclassified technical analysis of the security risks of emanations from computer monitors in 1985 [1]. The next important publication regarding video emanations came from Markus Kuhn nearly 20 years later in 2003 [6].

1.2 Related Work

With the notable exception of van Eck and Kuhn, there is almost no research on the topic of compromising emanations from video displays in the open literature. This is worrying since the phenomenon is real and could easily be exploited by an adversary.

1.2.1 Work of Wim van Eck

Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk? [1] was the first publicised research on the topic of emanations from video monitors. Wim van Eck describes the similarities between contemporary monitors and black and white TV sets. He explains that these could be exploited to build a cheap eavesdropping device.

The document describes usage of a modified black and white commercial television receiver to conduct a very successful practical attack. Furthermore, a demonstration is done outside the laboratory in real world conditions. Van Eck proves that the attack is a viable security thread. He continues discussing possible sources of the emissions and ways of defending against such a attacks.

However, there are some problems with his implementation:

- The modification of the TV receiver, although inexpensive, requires advanced specialised knowledge to undertake.
- The fact that the device needs constant manual adjustments to keep the oscillators in the receiver and transmitter in sync makes it difficult to use in practice.
- Technology has changed dramatically since the publishing of the paper. The attack will no longer work on modern monitors due to the variety of video modes available which now differ significantly from broadcast television.
- No room for further automated signal processing.

We can only speculate why there was no further research on this topic for decades to come. Possibly the main reason was that a modified TV receiver will no longer work with modern day video modes and commercial off-the-shelf narrowband receivers can't be used for that purpose either. As Kuhn outlines, a researcher needed to have their hands on a very special military grade wideband receivers that are expensive and have export restrictions [6]. However, the project presented in this dissertation aims to solve this problem by using an affordable software-defined radio platform for receiving the emanations from video displays.

1.2.2 Work of Markus Kuhn

Markus Kuhn was able to improve on the results achieved by van Eck with his *Compromising emanations: eavesdropping risks of computer displays* [6]. In the technical report, Kuhn analyses in a very mathematically detailed manner the properties of the waves emitted and their possible emitting circuitry. He describes the possible equipment an attacker may need to intercept and process the signal. He conducts experiments and constructs a system for real time monitoring using an FPGA board, a specialised wideband AM radio receiver and an off-the shelf VGA video monitor.

However there are a few caveats:

- A researcher will need access to very expensive, export restricted equipment in order to repeat his experiments. This equipment also often lacks the mobility required for a practical attack.
- As with van Ecks' system, Kuhn's solution still requires manual synchronisation.
- Digital signal processing for improving reception of weak signals (like time averaging) was not attempted.
- An attacker will need to know the exact video mode the target is using.

The report also mentions possible ways of modulating hidden messages into the signal. Kuhn also discusses ways of defending against such attacks using hardware and software solutions. He demonstrate automatic character recognitions. The report also includes experiments with optical eavesdropping of CRT displays.

Overall his real-time monitoring system improves on van Eck's one by supporting a variety of modern video modes. Thus Kuhn was able to prove that the threads outlined almost two centuries ago are still valid. In a later paper [7] he focuses on digital displays and explains that the same attack vector could also be used in that context.

The presented project is inspired by Kuhn's work. His report simplifies further research into the topic by giving a very detailed and mathematical description of the emanated signal and its properties. This was an invaluable starting point proving that a software-defined radio implementation could be viable.

1.3 Achieved Goals and Motivation

The project aims to raise awareness of the potential security implications of the compromising emanations from video monitors. It implements the

principles outlined in the related work section 1.2 using a completely new approach: utilising a software-defined radio receiver. It also combines them into a single portable software library. In its basic form it allows tuning to a video signal by manually controlling the vertical and horizontal synchronisation similar to the implementation that van Eck and Kuhn have used in their practical demonstrations.

The system builds on top of the existing research by having achieved the following additional goals that were never done before:

- Uses affordable unmodified off-the-shelf equipment portable enough to simplify practical attacks.
- Almost no specialised experience is necessary to operate the system.
- No prior knowledge of the target machine video parameters is required. The system can automatically estimate them remotely in real time.
- Reception of weaker signals is possible due to additional digital signal processing.
- Eliminates the need for constant manual adjustments in order to keep the picture steady on the screen.
- Open source license aiding further research on the topic.

The source code and the latest pre-compiled binaries could be obtained from github.com/martinmarinov/TempestSDR. If user's computer is running OS X, Windows or Linux, a multiplatform statically precompiled Java jar file is available for starting up the system in a couple of clicks. [1]

The system supports a plug-in architecture that allows simplified development of device drivers for any software-defined radio frontend. Currently supported plug-ins:

- Pre-recorded file with raw IQ samples (Multiplatform support)

[1] Some device drivers might not be available on all operating systems. At time of writing the OS X binaries need to be compiled manually. Refer to the README for more information.

- Mirics S(Windows support)
- UHD (Linux and OS X support)
- ExtIO (Windows support)

The system comes with a Java GUI but the underlying library could be used independently by any other system as a shared or statically linked library. Furthermore the core is written in C and has no additional dependencies making it portable and easy to compile.

This report provides some background information on the physics behind the radio wave generation. It describes the basics of a Sradio receiver. It then explains the properties of the emanated signal and how to reconstruct it. It goes on describing the automatic remote estimation of video parameters. A practical attack is demonstrated afterwards showing that an attacker with no prior knowledge of the target video display can receive its signal from a distance using the system. It then outlines the architectural details of the library and the implementation details. Finally, a conclusion is done with some suggestions on further research.

However, the report does not discuss ways of limiting the amount of information leaked via such compromising emanations. This is beyond the scope of the project. The system, however, could be used for further research on the topic. It could possibly aid such attempts by serving as a rapid prototyping tool.

The report also does not claim that the experimental results presented are exhaustive. This is due to the fact that there is a wide range of software-defined radio front-ends that are supported, each of them having different characteristics. Therefore no attempt was done to characterise the hardware used for the experiments. The presented measurements are not provided as absolute value (i.e. voltages) but rather as relative (i.e decibels). The results will very much depend on the target video display as well. Some displays will emit strong signals that could be picked up tens of meters away, while others emit no detectable video emanations. Characterising receivers and

target video displays could be also left as a further research topic.

There are also some inherent notes about the system:

- The performance of the system depends on the speed of the CPU of the attacker. No GPU acceleration was implemented since it wasn't really necessary. Performance never dropped below 35 fps on an Intel Core i5 laptop without GPU acceleration.
- Difficult to compare its core performance to existing implementations due to the fundamental differences of the underlying architectures.
- Software-defined radio receivers have a lower sensitivity and higher noise figure due to lack of high quality analogue filters compared to specialised wideband receivers. There is also interference caused by their internal IC circuits or the USB/Ethernet/Power cables that connect them to the PC. However, Kuhn showed that in practice this could be greatly improved using suitable analogue preconditioning.

Chapter 2

Background

A reader should have knowledge of undergraduate level mathematics including calculus. Some of the equations and theorems are simply stated and then used. If the reader wants to learn more about their derivations, they need to look at a relevant textbook. Furthermore, it is assumed that the reader has a computer science background and understands the basic concepts behind object oriented programming and procedural programming. Familiarity with the C programming language and some basic knowledge of Java would be helpful as well. Understanding the basics of signal processing and Fourier theory is also desirable but not strictly required.

2.1 Signal Processing

The fact that changing currents in a wire induce currents in another nearby wire comes straight from Maxwell's equations [8]. A signal is just changing current and/or voltage in the time domain. Let's imagine we have a wire that we call wire A. We want to transmit a signal over it. We connect it to a signal generator that modulates some data $I(t)$, so that current across the wire at time t could be written as $x(t) = I(t)\sin(2\pi f_c t)$, where f_c is the carrier **frequency**. The argument of the sin, namely $2\pi f_c t$, is called the

phase of the wave. Because of Maxwell's equations, the changing current in wire A will also generate a circular changing magnetic field around itself.

Let's imagine that an adversary puts a wire B lying parallel to wire A. The changing magnetic field from wire A will generate a changing current in wire B. The adversary can measure the current in wire B as a function of time $\hat{x}(t)$. They will end up with $\hat{x}(t) = \hat{I}(f_c) \cdot I \cdot \sin(2\pi f_c t + \varphi(f_c))$ where $\hat{I}(f_c)$ is the attenuation of the peak amplitude and $\varphi(f_c)$ is the phase difference of the signal in wire B. Those particular values would depend on the physical properties of the system such as capacitance between the wires and their resistance. Having measured the carrier frequency f_c and by having some assumptions about the signal, an adversary could potentially make an estimate about $I(t)$. This is the source of the compromising emissions we observe – signals in wires unintentionally inducing currents in nearby wires, broadcasting some information about the original signal. In practice signals are more complex than just a single sin term, however Fourier analysis can help us.

The **Fourier transform** [9] allows us to see how the signal spectrum looks like in the frequency domain. This will decompose the function into a sum of sin and cos terms in the complex plane. The Fourier transform of $g(t)$ is defined as

$$G(f) = \mathcal{F}\left\{g(t)\right\}(f) = \int_{-\infty}^{\infty} g(t) \cdot e^{-2\pi j f t} \mathrm{d}t$$

and the inverse Fourier transform

$$g(t) = \mathcal{F}^{-1}\left\{G(f)\right\}(t) = \int_{-\infty}^{\infty} G(f) \cdot e^{2\pi j f t} \mathrm{d}f.$$

A sine wave of the form $x(t) = \sin(2\pi f_c t)$ will result in spikes $X(f) = \frac{1}{2i}[\delta(f - f_c) - \delta(f + f_c)]$ at $X(-f_c)$ and $X(f_c)$ since $X(f) = 0$ everywhere else. Therefore we can see that the Fourier transform basically gives an indication of the frequency of the source signal. Note that the **Dirac delta** function is defined as

$$\delta(t) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases}.$$

The Dirac delta functions have the so-called "sifting" property so that if multiplied with a function $g(t)$, it will **sample** out only a single value, namely

$$\int_{-\infty}^{\infty} g(t)\delta(t-T)\mathrm{d}t = g(T).$$

We will also encounter the "Dirac comb" defined as:

$$\text{Ш}_T(t) = \sum_{k=-\infty}^{\infty} \delta(t-kT) = \frac{1}{T}\text{Ш}\left(\frac{t}{T}\right)$$

which is basically an infinite number of Dirac delta functions repeating at regular intervals of T. We can therefore "sample" a function $x(t)$ at times T by multiplying it with a Dirac comb. This will result in discrete samples being obtained from $x(t)$ at regular intervals of T i.e. we will obtain $\ldots, x(-2T), x(-T), x(0), x(t), x(2T), \ldots$.

We also need to mention **convolution**, denoted by the operator $*$. The convolution of two functions $x(t)$ and $g(t)$ is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\mathrm{d}\tau.$$

To illustrate how convolution works, consider convolving a function $x(t)$ with a Dirac delta

$$\big(\{x(t)\} * \{\delta(t-T)\}\big)(t) = \int_{-\infty}^{\infty} x(\tau)\delta(t-\tau-T)\mathrm{d}\tau = \int_{-\infty}^{\infty} x(\tau)\delta(\tau-(t-T))\mathrm{d}\tau = x(t-T) \quad (2.1)$$

where we used the fact that the Dirac delta function has an even symmetry $\delta(t) = \delta(-t)$. This results in a new function $x(t-T)$ which is a time shifted copy of $x(t)$ with a shift of T. Convolving with a Dirac comb could be shown to produce infinite copies of $x(t)$ at regular intervals T which are basically summed together.

The term **lowpass** means a lowpass filter. This is a filter that acts on a

signal, and that allows components of the signal with frequencies $f \leq f_{\max}$ to pass through, attenuating the rest to 0. A **Highpass** filter allows frequencies $f \geq f_{\min}$ to pass through intact, attenuating the rest to 0. A **bandpass** filter is a combination of a lowpass and a highpass filter, allowing only frequencies $f_{\min} \leq f \leq f_{\max}$, attenuating the rest to 0. A **band-limited** signal is a signal which doesn't have any frequency components $f > f_{\max}$ or $f < f_{\min}$. Such a signal would remain unchanged after applying a bandpass filter to it with limits $f_{\min}$ and $f_{\max}$. This is used for filtering out useful signals from other interferences that are not part of the band-limited signal.

2.2 IQ Sampling

A simple analogue to digital converter will measure the voltage at a wire by producing f_{s} samples per second (called **sampling rate**). This means that it will produce a new sample every $t_{\mathrm{s}} = \frac{1}{f_{\mathrm{s}}}$ seconds. It converts an analogue signal a.k.a. continuous function $x(t)$ into a series of samples. This could be written mathematically as a multiplication with a Dirac comb

$$\hat{x}(t) = \sum_{k=-\infty}^{\infty} x(t)\delta(t - kT). \tag{2.2}$$

If we want to detect a signal containing frequency components from 0 up to $f_{\max}$, then according to the Nyquist sampling theorem, we need to sample with a sample rate $f_{\mathrm{s}} > 2f_{\max}$. This usually gets impractical when $f_{\max}$ gets large.

However, if we are only interested in a band-limited signal that lies from $f_{\min}$ up to $f_{\max}$, there is another solution. We can have a local oscillator that runs at a frequency $f_{\mathrm{LO}} = \frac{f_{\min}+f_{\max}}{2}$. The oscillator generates a signal $A_{\mathrm{LO}} \cdot \cos(2\pi f_{\mathrm{LO}} t)$. The output from the local oscillator is multiplied with the incoming **RF** (radio frequency) baseband signal (a.k.a. mixing). Say the RF signal is $x(t) = A(t)\cos(2\pi f_{\mathrm{c}} t)$. It has a frequency of f_{c} and phase of $2\pi f_{\mathrm{c}} t$. For simplicity, let's put $A_{\mathrm{LO}} = 2$. After the mixing stage, our resulting signal

will be

$$x(t) \cdot A_{\mathrm{LO}} \cos(2\pi f_{\mathrm{LO}} t) = A(t)\cos(2\pi f_{\mathrm{c}} t) \cdot 2\cos(2\pi f_{\mathrm{LO}} t) = \\ = A(t)\cos\big(2\pi(f_{\mathrm{c}} + f_{\mathrm{LO}})t\big) + A(t)\cos\big(2\pi(f_{\mathrm{c}} - f_{\mathrm{LO}})t\big). \quad (2.3)$$

This results in a linear combination of two signals with frequencies called heterodyne frequencies – the sum and the difference between the local oscillator and the incoming frequency. A bandpass filter could be set up at an **IF** (intermediate frequency) f_{IF}. If any of these heterodyne frequencies falls within the bandpass of the filter, the signal can go for further processing.

However, we encounter a problem: cos is an even function. Imagine we now have another RF baseband signal, namely $\hat{x}(t) = A(t)\cos\big(2\pi(2f_{\mathrm{LO}} - f_{\mathrm{c}})t\big)$. After mixing with the local oscillator, the resulting signal would be $A(t)\cos\big(2\pi(f_{\mathrm{LO}} - f_{\mathrm{c}})t\big)$ which is equivalent to what we obtained for $x(t)$ in (2.3) for the difference $f_{\mathrm{LO}} - f_{\mathrm{c}}$ signal. So we can't distinguish $\hat{x}(t)$ from $x(t)$ if we decide to only filter out the difference part. This is called signal imaging. It could be shown that a signal $\tilde{f}(t) = A(t)\cos\big(2\pi(-f_{\mathrm{c}} - 2f_{\mathrm{LO}})t\big)$ will in turn generate a sum $A(t)\cos\big(2\pi(-f_{\mathrm{LO}} - f_{\mathrm{c}})t\big)$ which again results in imaging if we decide to keep the summation term $f_{\mathrm{LO}} + f_{\mathrm{c}}$ from (2.3).

This method can preserve the amplitude of the wave and is widely used in AM radio reception (in conjunction with some image rejection techniques as well). However, it is obvious from the problem outlined above that the method is not sufficient for uniquely detecting the phase of the incoming wave. A solution to this problem is to multiply the incoming RF signal with a phase shifted version of the local oscillator at an angle of 90° in parallel with the mixing from (2.3). This means multiplying by $-A_{\mathrm{LO}} \cdot \sin(2\pi f_{\mathrm{LO}} t)$. After this secondary mixing stage, we would get

$$-x(t) \cdot A_{\mathrm{LO}} \sin(2\pi f_{\mathrm{LO}} t) = -A(t)\cos(2\pi f_{\mathrm{c}} t) \cdot 2\sin(2\pi f_{\mathrm{LO}} t) = \\ = -A(t)\sin\big(2\pi(f_{\mathrm{c}} + f_{\mathrm{LO}})t\big) + A(t)\sin\big(2\pi(f_{\mathrm{c}} - f_{\mathrm{LO}})t\big). \quad (2.4)$$

Now let's put $\varphi(t) = 2\pi(f_{\mathrm{c}} - f_{\mathrm{LO}})t$, and we only take the terms that contain

φ in (2.3) and (2.4), namely the difference terms using a bandpass filter of width $\frac{f_\mathrm{s}}{2}$. Then we can define

$$I(t) = A(t)\cos(\varphi(t)) \quad \text{and} \quad Q(t) = A(t)\sin(\varphi(t)). \tag{2.5}$$

Sampling these continuous signals with an analogue to digital converter will result in discrete values at times t_{f_s}. Let's put $A_n = A(nt_{f_\mathrm{s}})$ and $\varphi_n = \varphi(nt_\mathrm{s})$. Then we can define our resulting discrete samples as

$$I_n = A_n\cos(\varphi_n) \quad \text{and} \quad Q_n = A_n\sin(\varphi_n). \tag{2.6}$$

These I and Q samples form the basic principles of software-defined radio. I and Q stand for in-phase and quadrature phase [10]. These terms refer to the cos and $-\sin$ oscillator phases used for mixing. After the IQ samples are obtained, they are received by a PC with a sample rate of f_s.

They allow monitoring frequencies from $-\frac{f_\mathrm{s}}{2}$ to $\frac{f_\mathrm{s}}{2}$. The Nyquist sampling theorem would allow a maximum frequency span from 0 to $\frac{f_\mathrm{s}}{2}$ for signals sampled with real values. This would cause the imaging we discussed before. IQ sampling improves on that using complex numbers effectively doubling the range.

Once IQ samples are obtained by software, further analysis of the phase, frequency and amplitude could be done allowing for any signals that fit within the bandwidth from $-\frac{f_\mathrm{s}}{2}$ to $\frac{f_\mathrm{s}}{2}$. The following identities could be used to obtain them for a sample n (ω_n is the frequency)

$$A_n = \sqrt{I_n^2 + Q_n^2} = \sqrt{A_n^2 \cdot (\cos^2(\varphi_n) + \sin^2(\varphi_n))} \tag{2.7a}$$

$$\varphi_n = \tan^{-1}\left(\frac{Q_n}{I_n}\right) = \tan^{-1}\left(\frac{\sin(\varphi_n)}{\cos(\varphi_n)}\right) \tag{2.7b}$$

$$\omega_n = \frac{\mathrm{d}\varphi_n}{\mathrm{d}t} = \varphi_n - \varphi_{n-1}. \tag{2.7c}$$

Refer to Figure 2.1 for geometrical representation of the outlined properties.

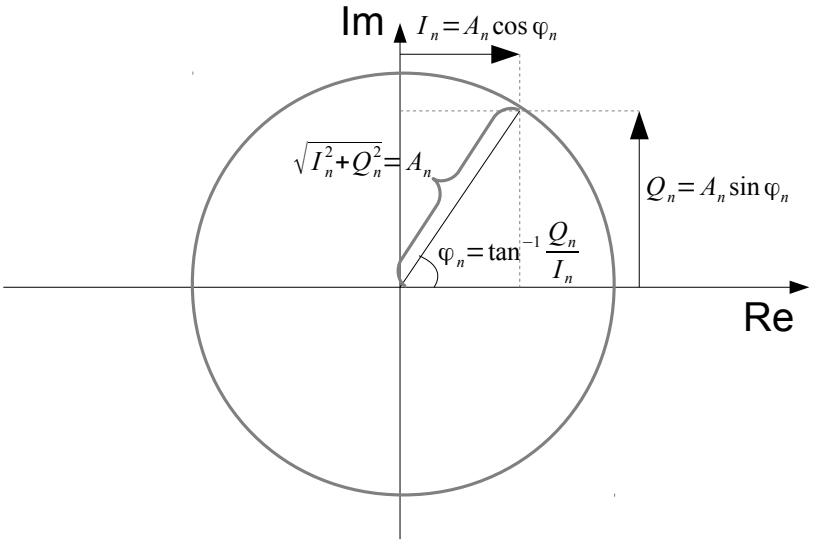


Figure 2.1: Graphical representation of an IQ sample n in the complex plane. All points with the same amplitude A_n will lie on the a circle showed in gray. The angle that the sample vector makes with the real axis is the instantaneous phase φ of the sample.

Chapter 3

Methodology

3.1 Analogue Video Signals

Almost all contemporary video monitors are raster based. The image is transferred from the video controller to the display in scanlines that occur at a specific rate. Each of these scanlines contains a number of pixels which are continuously encoded as a time varying signal. This signal is generated internally in the video controller by an oscillator which runs at the pixel clock rate. The analogue signal is thereafter multiplied by the pixel intensities at the specific time.

Let's assume that the signal started transmitting at time $t = 0$, and each video frame contains y_t scanlines, each of which contains x_t pixels. The frequency with which frames are being generated is f_v frames per second. The duration of transmission of each individual pixel is

$$t_\mathrm{p} = \frac{1}{x_\mathrm{t} \cdot y_\mathrm{t} \cdot f_\mathrm{v}}. \tag{3.1}$$

Note that at time t, frame number $n(t)$ started transmission where

$$n(t) = \lfloor t f_\mathrm{v} \rfloor. \tag{3.2}$$

We can assume that the top left corner of a frame has coordinates $(0, 0)$, a pixel at position (x, y) in frame $n(t)$ will start to be transmitted at time

$$T_{(x,y)} = (n(t)x_{\mathrm{t}}y_{\mathrm{t}} + yx_{\mathrm{t}} + x)t_{\mathrm{p}} \tag{3.3}$$

and will finish transmission before time

$$T_{(x+1,y)} = T_{(x,y)} + t_{\mathrm{p}} = (n(t)x_{\mathrm{t}}y_{\mathrm{t}} + yx_{\mathrm{t}} + x + 1)t_{\mathrm{p}} \tag{3.4}$$

at which the next pixel will start transmitting.

In practice x_{t} and y_{t} are determined by the screen resolution and f_{v} is simply the screen refresh rate. For a typical screen resolution of $width \times height$, it is true that $x_{\mathrm{t}} \geq width$ and $y_{\mathrm{t}} \geq height$. The reason is that video signals tend to have additional blanking intervals. This means more pixels are transmitted than what is in the active video region. This gives opportunity for the receiving monitor to synchronise its internal clock, calibrate its colour levels or in case of CRT, allow enough time for the electron beam to return to the beginning of the next line on the screen. The synchronisation timings for personal computers have been standardised by Video Electronics Standards Association (VESA) [11].

In order to decode an individual pixel, the receiving monitor has its own internal oscillator. It locks it to the pixel rate of the incoming signal either via an external clock source or using the blanking intervals. Once it receives the signal for an individual pixel, its amplitude (or binary content in case of digital signal) will correspond to the intensity. This allows the monitor to display the video in real time. If multiple colours are desired, they can be transmitted separately on different wires in the same fashion.

3.2 Generated Radio Wave

3.2.1 Analogue Signal Equation

Let's assume the discrete pixels in an analogue video signal have intensities v_i ($i \in \mathbb{Z}$) and are being transmitted for duration $t_\mathrm{p} = \frac{1}{x_\mathrm{t} \cdot y_\mathrm{t} \cdot f_\mathrm{v}}$ from (3.1). Let's also assume that the shape of the pixel is $p(t)$ where $p(t) = 0$ for $|t| \gg \frac{t_\mathrm{p}}{2}$. We know that pixel i starts transmitting at time $t_i = it_\mathrm{p}$ (if we assume that that pixel 0 was transmitted at $t = 0$). Also the amplitude of the signal of the transmitted pixel is linearly dependant on its intensity v_i. Then the resulting video signal in the time domain will have the form

$$\tilde{v}(t) = \sum_{i=-\infty}^{\infty} v_i p(t - it_\mathrm{p}) \tag{3.5}$$

which is a continuous function. (3.5) can be rewritten as a convolution with a Dirac delta function

$$\tilde{v}(t) = p(t) * \left(\sum_{i=-\infty}^{\infty} v_i \cdot \delta(t - it_\mathrm{p}) \right) = p(t) * \hat{v}(t) \tag{3.6}$$

where

$$\hat{v}(t) = \sum_{i=-\infty}^{\infty} v_i \cdot \delta(t - it_\mathrm{p}). \tag{3.7}$$

Note that this results in infinitesimally short (in time) spikes at values exact integer multiples of t_p with amplitudes weighted to the pixel intensities at that time. We can view (3.6) as the pixel shape being repeated at intervals of t_p modulated by v_i.

3.2.2 Sampling

Note that (3.7) looks like the result of a continuous signal being sampled at discrete points in time t_p apart. If we call this hypothetical continuous signal

$v(t)$, then we can rewrite the equation so that it reads

$$\hat{v}(t) = \sum_{i=-\infty}^{\infty} v(t) \cdot \delta(t - it_{\mathrm{p}}) = v(t) \sum_{i=-\infty}^{\infty} \delta(t - it_{\mathrm{p}}) = v(t) \cdot \mathrm{Ш}_{t_p}(t). \quad (3.8)$$

The requirement is that if $v(t)$ is sampled at discrete intervals it_{p}, it should be equal to the corresponding pixel values i.e. $v(0) = v_0$, $v(t_{\mathrm{p}}) = v_1$, $v(2t_{\mathrm{p}}) = v_2$, etc. This would mean that we can precisely obtain the samples from the continuous function by applying the series of Dirac delta functions. However, now we would also like to be able to obtain the continuous signal by only having the sampled values v_i.This means that we need a unique perfect reconstruction i.e. to be able to obtain all v_i from $v(x)$ and $v(x)$ from all v_i. Under the assumption that $v(t)$ is band-limited so that $\mathcal{F}\big(v(t)\big)(f) = 0$ for $|f| < \frac{t_{\mathrm{p}}}{2}$, the condition is satisfied by the Whittaker–Shannon interpolation formula [12]. It states that the continuous signal could be uniquely reconstructed using sinc interpolation. If we apply it to our v_i samples, it yields

$$v(t) = \sum_{k=-\infty}^{\infty} v_k \cdot \mathrm{sinc}\left(\frac{t}{t_{\mathrm{p}}} - k\right) \quad (3.9)$$

where we use the definition $\mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$ .We can verify that when $t = it_{\mathrm{p}}$, then $\mathrm{sinc}(i - k)$ will always yield 0 except for when $i = k$ in which case it will be 1. Therefore we have proved that $v(it_{\mathrm{p}}) = v_i$.

3.2.3 Repetitions

Now let's revise some mathematical identities. According to the convolution theorem [13]

$$\mathcal{F}\{g \cdot h\} = \mathcal{F}\{g\} * \mathcal{F}\{h\} \quad (3.10)$$

which reads: the Fourier transform of a multiplication of two functions is equivalent to the Fourier transforms of the individual functions convolved

together. And vice versa

$$\mathcal{F}\{g * h\} = \mathcal{F}\{g\} \cdot \mathcal{F}\{h\}, \tag{3.11}$$

The convolution of the Fourier transform of two functions is equivalent to the individual Fourier transforms of the two functions multiplied together. It could be also shown that

$$\mathcal{F}\{\text{Ш}_k(t)\} = \mathcal{F}\left\{\sum_{n=-\infty}^{\infty} \delta(t - nk)\right\} = k^{-1} \sum_{n=-\infty}^{\infty} \delta\left(t - k^{-1}n\right) = k^{-1}\text{Ш}_{k^{-1}}(t) \tag{3.12}$$

using the last three identities, we can write the Fourier transform of (3.6) as

$$\tilde{V}(f) = \mathcal{F}\{p(t) * \hat{v}(t)\} = \mathcal{F}\left\{p(t) * \left(v(t) \cdot \text{Ш}_{t_p}(t)\right)\right\} = \frac{1}{t_\text{p}} P(f) \cdot \left[V(f) * \text{Ш}_{t_p^{-1}}(f)\right] \tag{3.13}$$

where $P(f)$ is the Fourier transform of $p(t)$, $\tilde{V}(f)$ is the Fourier transform of $\tilde{v}(t)$ and $V(f)$ is the Fourier transform of $v(t)$. This simply means that the signal spectrum $V(f)$ repeats at regular intervals throughout the radio spectrum with a frequency of $\frac{1}{t_\text{p}} = x_\text{t} \cdot y_\text{t} \cdot f_\text{v}$. The intensities of the emission at the different frequencies is determined by the shape of the pixel $p(t)$.

For example, for a resolution of 800×600 @ 75 fps with $x_\text{t} = 1056$ px, $y_\text{t} = 628$ lines and $f_\text{v} = 75$ Hz , we would have a frequency of $\frac{1}{t_\text{p}} = 1056 \times 628 \times 75 \simeq 49.74$ MHz. This means that we would expect to find such a signal centred at DC, 49.74 MHz, 99.48 MHz, 149.2 MHz, etc.

In order to obtain the full video signal, we need to know what is the minimum rate at which we need to sample the baseband. We saw from (3.13) that $V(f)$ repeats at the regular intervals. As a consequence from the Whittaker–Shannon interpolation formula that we used to construct $v(t)$, we know that $V(f)$ is band limited so that $V(f) = 0$ for all $|f| \geq \frac{1}{2t_\text{p}}$. Therefore we need to have a receiver with a sampling rate of at least $\frac{1}{t_\text{p}}$.

3.2.4 Digital Video Signals

The analysis above was aimed at analogue signals such as the ones transmitted via VGA. However the majority of contemporary laptops and similar devices don't simply modulate the pixel intensities as analogue voltage amplitudes. Instead, they transmit each pixel as digital bits using FPD-Link (Flat Panel Display Link) to transmit LVDS (Low-voltage differential signalling) signal. For each pixel, 7 bits are transmitted over each of the three wires that determine the resulting RGB colour.

Therefore each bit is transmitted for a duration of $t_\mathrm{b} = \frac{1}{x_\mathrm{t} \cdot y_\mathrm{t} \cdot f_\mathrm{v} \cdot n_\mathrm{b}} = \frac{t_\mathrm{p}}{n_\mathrm{b}}$ where n_b is the number of bits per pixel which in case of FPD-Link is $n_\mathrm{b} = 7$. This represents a bit stream of values c_k ($k \in \mathbb{Z}$). We defined the k-th value being the ($k \mod n_\mathrm{b}$) bit of the binary number that is used to represent the analogue pixel intensity of pixel number $\left\| \frac{k}{n_\mathrm{b}} \right\|$ (remember, the intensity of pixel number i is denoted as v_i).

Therefore, we can write the resulting analogue signal as

$$\tilde{v}(t) = \sum_{k=-\infty}^{\infty} c_k b(t - k t_\mathrm{b})$$

where $b(t)$ is the shape of a digital bit where $b(t) = 0$ for $|t| \gg \frac{t_\mathrm{b}}{2}$.

Now assuming $c(t)$ is the continuous version of c_k as in the previous section, we can do the same Fourier analysis on this signal, similar to (3.13) to obtain

$$\mathcal{F}\left\{ b(t) * \left(c(t) \cdot \text{Ш}_{t_b}(t) \right) \right\} = \frac{1}{t_\mathrm{b}} B(f) \cdot \left[C(f) * \text{Ш}_{t_b^{-1}}(f) \right]. \tag{3.14}$$

This means that we will also get repeating signal throughout the spectrum, this time repeating with a frequency of $\frac{1}{t_\mathrm{b}} = x_\mathrm{t} \cdot y_\mathrm{t} \cdot f_\mathrm{v} \cdot n_\mathrm{b}$. This is n_b higher than the calculated repetition frequency for analogue signals. For example, Markus Kuhn's laptop was running LVDS with $n_\mathrm{b} = 7$ at a video mode of 800×600 @ 75 with $x_\mathrm{t} = 1056$ px, $y_\mathrm{t} = 628$ lines and $f_\mathrm{v} = 75$

Hz. He reports receiving a harmonic emanation at about 350 MHz[1] [7]. We can see that this agrees with the theoretical predictions since for his case $\frac{1}{t_\mathrm{b}} = 1056 \cdot 628 \cdot 75 \cdot 7 \simeq 348.2$ MHz.

However, the signal is now wider. We would need a receiver that can pick up $|f| \geq \frac{1}{2t_\mathrm{b}} \equiv \frac{n_\mathrm{b}}{2t_\mathrm{p}}$ in order to reconstruct individual bits. In practice, we can still use the same approach that we used for analogue video. We can have a receiver that can pick up $|f| \geq \frac{1}{2t_\mathrm{b}}$. This is essentially equivalent to applying a lowpass filter – we will no longer be able to distinguish individual bits. Instead we will obtain an averaged out value for each pixel that depends on the underlying binary data that is used to represent its intensity. The variety of colours available with significantly different bit patterns means that we can still extract a lot of visual information from the reconstructed image.

3.3 Reception

3.3.1 Theory

In order to receive a copy of the video signal, we will need to apply a bandpass filter centred at a multiple of the pixel frequency. The width of the bandpass needs to be the same as the band limited video signal we want to eavesdrop. Then we need to generate an internal clock that runs at the pixel rate and use that to obtain the estimated pixel intensities. Luckily these steps are already done in a typical AM (Amplitude Modulation) receiver. Unfortunately the signal bandwidth is much wider than what a typical off-the-shelf AM receiver can handle.

However some software-defined radio receivers can provide the required bandwidth. The Ettus Research USRP B200 can provide more than 50 MHz of realtime bandwidth which covers what is required to eavesdrop most video

[1] His actual hardware is transmitting the pixel two at a time, resulting in a factor of $\frac{1}{2}$ being applied to the end result. Therefore the first harmonic he could actually pick up lies at 175 MHz.

signals. Software-defined radios provide the radio samples as a quadrature vector which angle relates to the instantaneous phase in relation to the internal hardware oscillator of the hardware. The size of the vector is proportional to the received amplitude for each sample (refer to Section 2.2 for more information). For AM reception, we do not need the phase information since the video signal is being transmitted with a constant frequency. Therefore we can make a simple AM demodulator by taking the length of the vector of each sample. When our digital sampling rate matches the pixel rate, each sample will contain an estimate that relates to the average pixel intensity between time t_i and t_{i+1}. For digital signals, it would be related to the bit pattern of the binary value (although this relation may not be unique) that represents the intensity for the current pixel.

3.3.2 Practice

The VESA standard specifies an allowed frequency tolerance for f_v of 0.5%. Therefore due to hardware limitations, we might not be able to accurately tune the receiver sampling rate with enough accuracy to match the transmitted pixel rate. This means that some resampling needs to be done in software in order to create a new digital signal that matches as close as possible the transmitted pixel rate in which each sample corresponds to a pixel.

It is also true that we might be able to recover a lot of information from a signal even if our receiver is not capable of sampling the full width of the video spectrum. This means we will only receive a of $v(f)$ with a bandpass filter applied to it. In this case the digital resampling could interpolate the received samples to fit into multiple pixels. Therefore it is useful to introduce a measurement κ of how accurate we can reconstruct pixels of $v(t)$. This measurement can be the number of input samples that have gone into constructing a single output pixels. Therefore

$$\kappa = f_\mathrm{s} \cdot t_\mathrm{p} = \frac{f_\mathrm{s}}{x_\mathrm{t} \cdot y_\mathrm{t} \cdot f_\mathrm{v}} \tag{3.15}$$

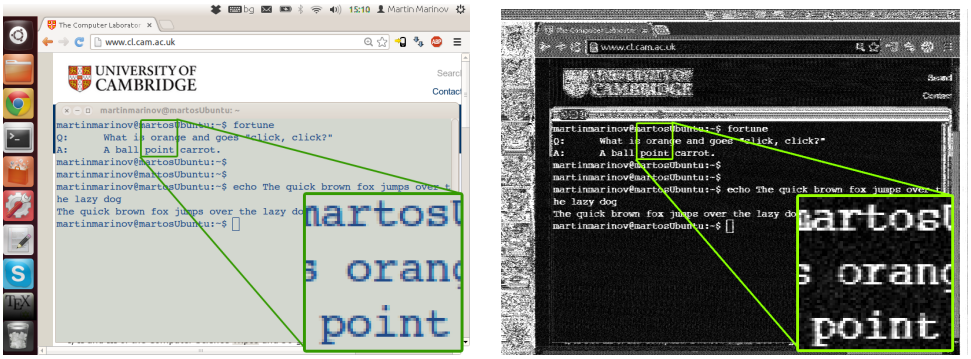
where as before, f_s is the receiver sampling rate. If $\kappa \geq 1$, then we can deconstruct individual pixels. If $\kappa < 1$ we are looking at a band-passed version of the signal and consecutive reconstructed pixels will have some interdependencies. Please refer to Figure 3.1 for a visual comparison of different values of κ.

Another possible source of distortions is the usage of multiple wires to transmit colour information. These signals interfere with each other and the resulting pixel intensity becomes a complicated mix of the signals generated in these wires. Therefore colour information would be very difficult to obtain. Nevertheless the outlined strategy allows an eavesdropper to detect changes in colour since this would result in different signal amplitudes arriving at the receiver for each pixel.

3.4 Resolution and Framerate Detection

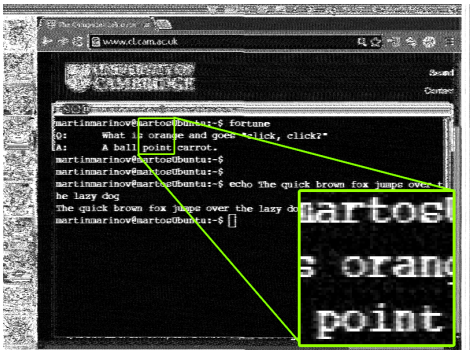
The outlined method for decoding the video stream relies on us having the exact values for x_t, y_t and f_v. This is not very practical for a real world attack. We would require the adversary not to have any knowledge of the target system whatsoever. Therefore we would need to infer all of these parameters remotely by exploiting some typical characteristics of a video signal. Namely the fact that the signal is periodic.

As we have already seen, a video signal consists of video frames with width x_t and height y_t. The speed at which such frames are transmitted is f_v. However most of the time any two consecutive frames would be identical. This would be the case if the victim is looking at static text on the screen. Therefore we can regard the transmitted signal as a repeating periodic signal of a single frame. Which means if we analyse the signal for repeating patterns we should be able to spot the repeating video frames in it and use that to estimate f_v.

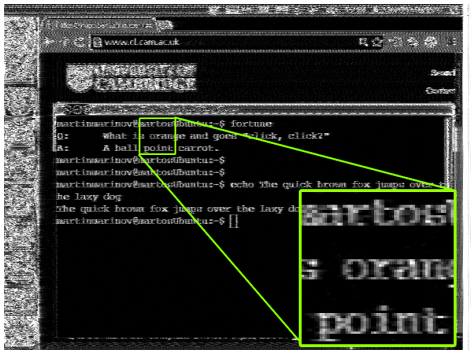


(a) Transmitted image

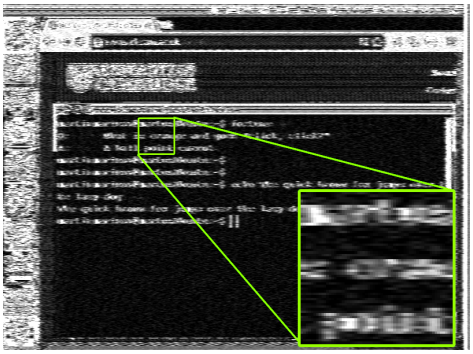
(b) $\kappa = 1.003$



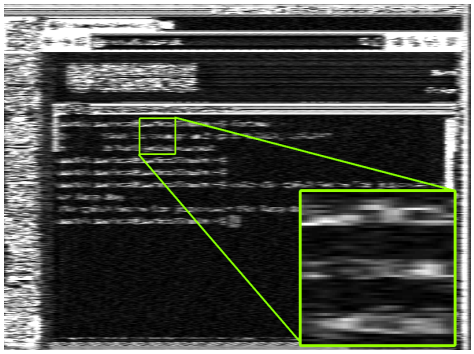
(c) $\kappa = 0.753$



(d) $\kappa = 0.502$



(e) $\kappa = 0.251$



(f) $\kappa = 0.125$

Figure 3.1: The transmitted image (a) is sent via a standard HDMI to VGA converter. The emanations are coming from the digital video signal. The resolution is 800×600 @ 60 with $x_\mathrm{t} = 1056$, $y_\mathrm{t} = 628$ and $f_\mathrm{v} = 60.11$. Each of the following screenshots represent an eavesdropped version from a distance of 1 m with the receiver (USRP B200) running at different sample rates: (b) 40 MHz, (c) 30 MHz, (d) 20 MHz, (e) 10 MHz and (f) 5 MHz

3.4.1 Introduction to Autocorrelation

A good way of doing so would be calculating the discrete autocorrelation [14] of the incoming signal which is defined as

$$R_{vv}(j) = \sum_{i=-\infty}^{\infty} v_i \bar{v}_{i-j}$$

where $\bar{v}$ is the complex conjugate of v and j is the **samples lag**. In our case v is the set of real samples, therefore $\bar{v} = v$. Basically the autocorrelation is a measure of the similarity of a signal with itself shifted by a lag. The higher the value of $R_{vv}(j)$ is, the more similar the function is at that lag. Therefore by analysing $R_{vv}(j)$, we would be able to spot any repeating patterns inside v_i.

Figure 3.2 shows the discrete autocorrelation of a video signal that was captured using the Mirics USB dongle at 8 MHz (i.e. $8,000,000$ samples per second) sampling rate. The vertical axis is logarithmic, containing the decibels of the autocorrelation i.e. $10\log_{10}(R_{vv})$. The horizontal axis contains the time lag in milliseconds. This was converted from the lag in samples j by $x = \frac{j}{f_s}$.

This particular example consists of $524,288$ samples which is 65.536 milliseconds of recording. This allows us to estimate the autocorrelation up to half of it, namely 32.768 milliseconds. The reason is because $R_{vv}(j) = -R_{vv}(j)$ due to symmetry. We can see a peak at 16.672 milliseconds which corresponds to a frequency of $f_\mathrm{v} = \frac{1}{0.016672} = 59.98$ Hz. This is our estimate for f_v. And it makes sense, 60 Hz is a common refresh rate for contemporary video displays.

3.4.2 Aliasing

We should be able to see the peak repeating again at $2 \times 16.672 = 33.344$ ms. This is analogous to comparing every two consecutive frames with every

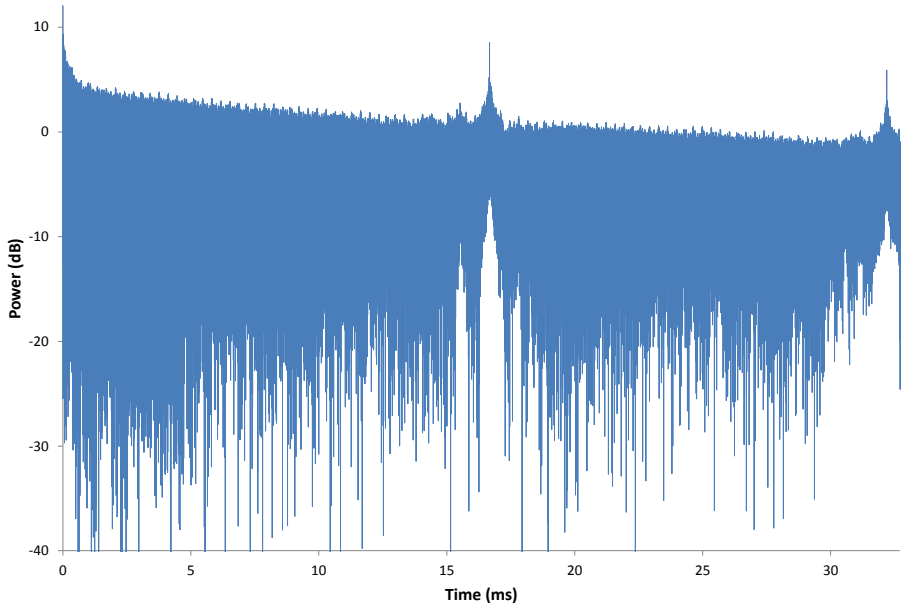


Figure 3.2: Autocorrelation of a signal

two other frames. However, as this falls just outside our window it would be seen as an alias at $65.536 - 33.344 = 32.192$ ms. The same applies for the third peak at $3 \times 16.672 = 50.016$ ms which would be comparing each block of three consecutive frames with the next three. This will alias at $65.536 - 50.016 = 15.52$ ms. This explains the second small peak we see on the left of the main one at 16.672 ms but we see that the difference in its power is already several decibels lower. Therefore any further aliasing induced would be even smaller.

It is therefore important to choose the number of samples to be used for the autocorrelation wisely in order to avoid aliasing. We need to have enough full frames in our data so that the autocorrelation can physically work. The minimum frequency we can get from an autocorrelation is $\frac{2s}{N}$ where N is the total number of samples that we are doing the autocorrelation on. It could

be easily shown that the second alias is

$$\hat{f}_2(f) = \frac{1}{\frac{2}{f_{\min}} - \frac{2}{f}} = \frac{1}{\frac{N}{f_s} - \frac{2}{f}} \tag{3.16}$$

where f is the frequency we are going to see the alias for. Therefore if we want to be able to pick up a repeating pattern of at least $f_{\min} = 50$ Hz, we would need $\frac{2s}{50}$ which in our case is $320,000$ samples. However having such a small number of samples if we observe a signal at 85 Hz we would discover a strong alias at $\left(\frac{320{,}000}{8{,}000{,}000} - \frac{2}{85}\right)^{-1} = 60.714$ Hz which would be very misleading and will look like a legitimate signal at 60.714 Hz.

Let us assume $f_{\text{low}} \leq f_v \leq f_{\text{hi}}$. In order to minimise aliasing, we can try to keep the second alias $\hat{f}_2(f_{\text{src}})$ outside this range. Therefore we need to pick the number of samples so that

$$f_{\text{low}} \geq \hat{f}_2(f_{\text{low}}) \equiv \frac{1}{\frac{N}{f_s} - \frac{2}{f_{\text{low}}}} \quad \text{or} \quad f_{\text{hi}} \leq \hat{f}_2(f_{\text{hi}}) \equiv \frac{1}{\frac{N}{f_s} - \frac{2}{f_{\text{hi}}}}$$

which means that

$$N \geq 3\frac{f_s}{f_{\text{low}}} \quad \text{or} \quad N \leq 3\frac{f_s}{f_{\text{hi}}}$$

and don't forget that in the same time we have the condition

$$N \geq 2\frac{f_s}{f_{\text{low}}}$$

so in the end, in order to eliminate the second alias in the desired region from f_{low} to f_{hi} in the autocorrelation, we would need to choose the number of samples to be

$$N \geq 3\frac{f_s}{f_{\text{low}}}. \tag{3.17}$$

To paraphrase this result, we will need to do the autocorrelation on at least 3 frames worth of samples for the lowest frequency in our range f_{low} to avoid second order aliasing interfering with our plot.

3.4.3 Number of Lines in a Frame

We were able to estimate f_{v} using an autocorrelation and eliminate aliased signals in the frequency window that we are looking to detect repetitions. However, we still have not estimated neither x_{t}, nor y_{t}. As a matter of fact we might not be able to measure x_{t} at all. The reason is that x_{t} might be a continuous signal and could potentially contain any number of pixels. However, each line of x_{t} pixels repeat y_{t} times in a frame which includes the blanking intervals. Therefore we can use their repeating nature of the blanking intervals to estimate y_{t} from the autocorrelation plot.

For this reason we need to zoom in to our plot. Refer to Figure 3.3 which is a zoomed out version of the figure we saw in the previous subsection. If we still have our allowed f_{v} window between f_{low} and f_{hi}, then we can assume that y_{t} also spans between h_{low} and h_{hi}. We would therefore expect to see a peak between $\frac{1}{f_{\mathrm{hi}} \cdot h_{\mathrm{hi}}}$ and $\frac{1}{f_{\mathrm{low}} \cdot h_{\mathrm{low}}}$ milliseconds. If we detect a peak at time t_{peak}, then this would correspond to $y_{\mathrm{t}} = \left\| \frac{1}{t_{\mathrm{peak}} \cdot f_{\mathrm{v}}} \right\|$ where $\|$ represents rounding to the nearest integer. Note that we should have already estimated f_{v}.

Our plot shows a peak at 0.017875 milliseconds. Since we already estimated $f_{\mathrm{v}} = 59.98$ Hz, then $y_{\mathrm{t}} = \left\| \frac{1000}{0.017875 \times 59.98} \right\| = 933$ lines. If we look at a list of VESA video modes, we can see that there is a conveniently close video mode that corresponds to $y_{\mathrm{t}} = 933$ and $f_{\mathrm{v}} = 59.98$. It is the 1440×900 @ 60 video mode with $x_{\mathrm{t}} = 1904$ pixels, $y_{\mathrm{t}} = 932$ lines and $f_{\mathrm{v}} = 60$ Hz. Therefore we could use the value of $x_{\mathrm{t}} = 1904$ for our estimate of the number of pixels in a row. This will keep the aspect ratio correct.

We should not worry about aliasing since the number of samples in the autocorrelation is much bigger than our range. We can see on the plot the autocorrelation values repeating for each two lines, each three lines, etc. This results in a number of peaks repeating with the rate at which individual lines repeat in the video signal.

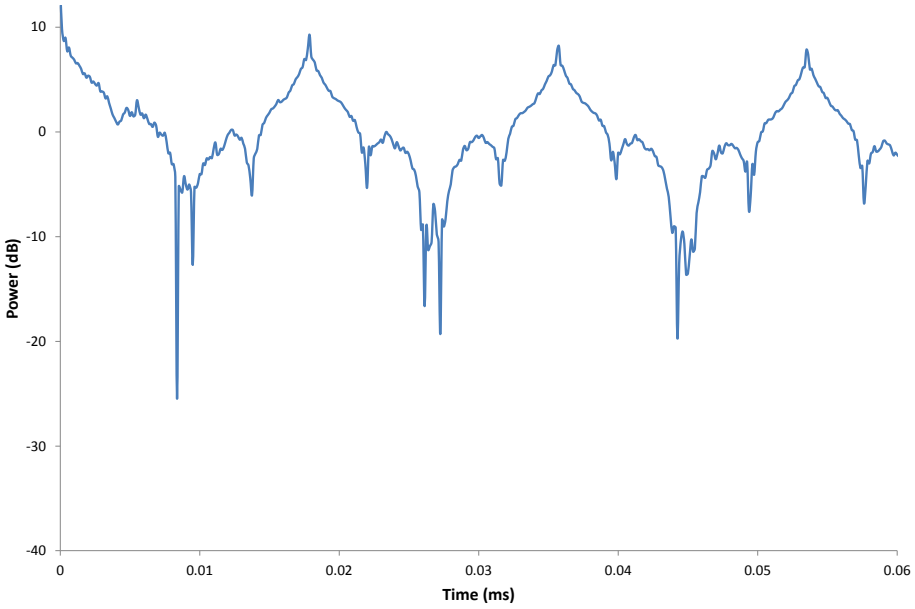


Figure 3.3: Zoomed in version of 3.2

3.4.4 Errors

There is a limit on how accurately one can determine f_{v} and y_{t} using the outlined autocorrelation method. This depends on the sampling rate of the device. The resolution of the autocorrelation is $\delta f = \frac{1}{f_{\mathrm{s}}}$. Therefore we can only say that if there is a peak at t_x in the autocorrelation, then we have a repeating component in the signal with a frequency $f_x \pm \delta f$.

The accuracy at which we can determine y_{t} depends both on the accuracy of f_{v} and t_{peak} and can vary. Furthermore y_{t} should be rounded to an integer (we can only display an integer number of lines on a PC screen) which introduces further errors. Our definition for y_{t} is

$$y_{\mathrm{t}} = \left\| \frac{1}{t_{\mathrm{peak}} \cdot f_{\mathrm{v}}} \right\| \pm (\delta y_{\mathrm{t}} + 0.5)$$

where 0.5 comes from the rounding. Also the uncertainties in t_{peak} and

f_v are δf since both are being read from the same autocorrelation plot in milliseconds. In order to do error propagation, we need to take the partial derivatives with respect to f_v and t_peak

$$\frac{\partial h}{\partial f_\mathrm{v}} = -\frac{1}{t_\mathrm{peak} \cdot f_\mathrm{v}^2} \quad \text{and} \quad \frac{\partial h}{\partial t_\mathrm{peak}} = -\frac{1}{f_\mathrm{v} \cdot t_\mathrm{peak}^2}.$$

So for the uncertainty in y_t, we have

$$\delta y_\mathrm{t} = \sqrt{\left(\frac{\partial h}{\partial f_\mathrm{v}}\delta f_\mathrm{v}\right)^2 + \left(\frac{\partial h}{\partial t_\mathrm{peak}}\delta t_\mathrm{peak}\right)^2} = \sqrt{\frac{1}{f_\mathrm{v}^2} + \frac{1}{t_\mathrm{peak}^2}} \times \frac{\delta f}{f_\mathrm{v} \cdot t_\mathrm{peak}}.$$

We can also observe that the larger the sample rate gets, the smaller δf and δt get.

Chapter 4

Practical attack

Before explaining how the software works internally, let's present a demonstration of a practical video eavesdropping attack. Its main aim is to show the ease with which such an attack could happen. In the meantime this will give an opportunity to explain the characteristics of the received signal and how they are exploited.

The demonstration is undertaken in controlled conditions. However, in order to make it as similar to a real-world attack as possible, we will assume the adversary has no knowledge of the victim's system. We will estimate the frequency at which the emission strength has the best strength. We will then analyse the signal to detect the resolution and refresh rate of the screen. We will afterwards lock onto the signal and try to recover the original video.

4.1 The Setup

The choice for a front-end for this demonstration is a USRP B200[1]. Depending on the particular requirements, an attacker might prefer mobility over accuracy and choose the smaller Mirics FlexiTV^{TM}MSi3101 SUSB Dongle.

[1]Refer to 5.1 Hardware for discussion on currently available devices.

However, for this demonstration we will attempt to obtain the highest possible resolution. Therefore we need an radio that is capable of obtaining a wide bandwidth. This makes the 32 MHz of Bandwidth that the USRP provides a much better choice than the 8 MHz available from the Mirics dongle.

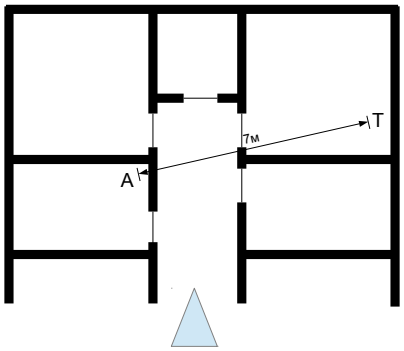
For an antenna, we will choose the portable MaxView active TV antenna. It is designed for digital television reception and has a built-in amplification. This is very useful for lifting signals above the USRP B200 noise floor. For better reception from longer distance, a suitable log-periodic antenna, an amplifier and a bandpass filter could be used for preconditioning. However, this makes the system less portable. However, the MaxView antenna is a cheaper choice and in the end did the required job.

You can view the equipment used by the adversary, the target and their mutual position on Figure **??**. We now take the place of the adversary that has set up their system, connected the UHD to the laptop and the antenna to the UHD and has launched the *JTempest.jar* executable.

4.2 The Attack

We should now see the GUI as in Figure 4.2. In order to start the USRP B200, we need to go to *File* and choose *Load USRP (via UHD)*. We can enter the command *–rate=8000000* which will set the sampling rate to 8 MHz. If no errors occur, we will see the "Start" button becoming active. Once we press on the button, the system will start processing the data sampled by the USRP.

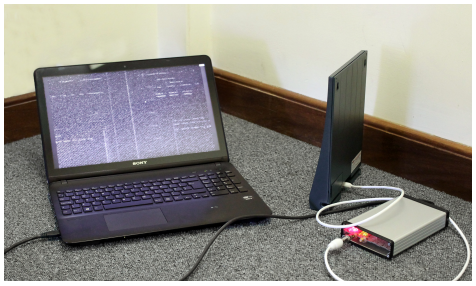
If we knew the resolution, frame-rate and the frequency at which the emanations from the target occur, we could enter them directly into the GUI and the story would end here. However, since we don't, we need to estimate them. That's why we started with a low sampling rate (8 MHz). Such mode will not be good for actual eavesdropping since it will not provide the full resolution, however this will allow for improved interactivity and for faster autocorrelation results.



The floor plan

The hallway (triangle)

Target (T)



Adversary (A)

Figure 4.1: The figure represents the set-up of the attack. The floor plan shows the position of the attacker and the target. The distance between them is 7 m and the signal passes through two solid walls. The target is a standard PC, connected to a normal VGA monitor. The adversary uses a laptop, a USRP B200 and a MaxView TV antenna. All is inexpensive off-the-shelf equipment. The set-up also allows for great mobility.

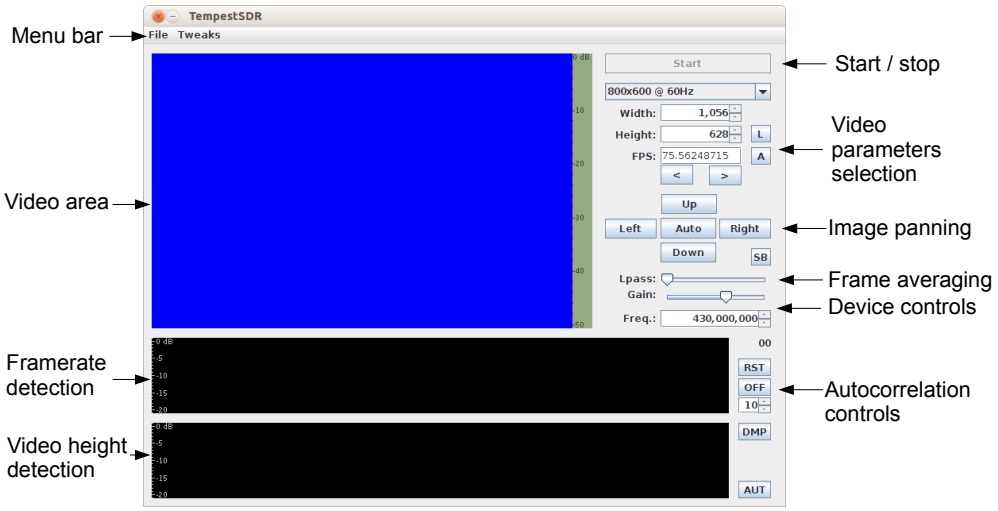


Figure 4.2: The interface of the program. The "A" button will enable the frame-rate tracking feature. The "Auto" button will attempt to detect synchronisation regions. The "AUT" button will try to automatically detect the resolution and line rate (video height). The scale on the right of the video shows in real-time the grey-scale values that map to a particular sample intensity.

First, we should estimate what is the frequency at which the target is broadcasting with highest power. To do that we need to keep an eye on the autocorrelation plot and keep changing the frequency until a promising signal appears. There would be a lot of background signals so this would be challenging. However, most video displays run at known refresh rates of 60 Hz, 75 Hz or 71 Hz. If we spot a signal near these values, then we know we are on to something. Figure 4.3 shows that there might be a possible signal onto the current frequency of 125 MHz.

In order to set the video parameters to the frequencies in those spikes, we just need to click with the mouse on top of them. We can inspect the frequency at a certain position by hovering the mouse on top. Once we select the best candidate for the refresh rate, we can now choose the best candidate for the line rate as well. This defines the height of the video frame. Note that we can try the "AUT" button which will simply choose the highest values of the two plots. However, when the signal to noise ratio is low, other repeating signals

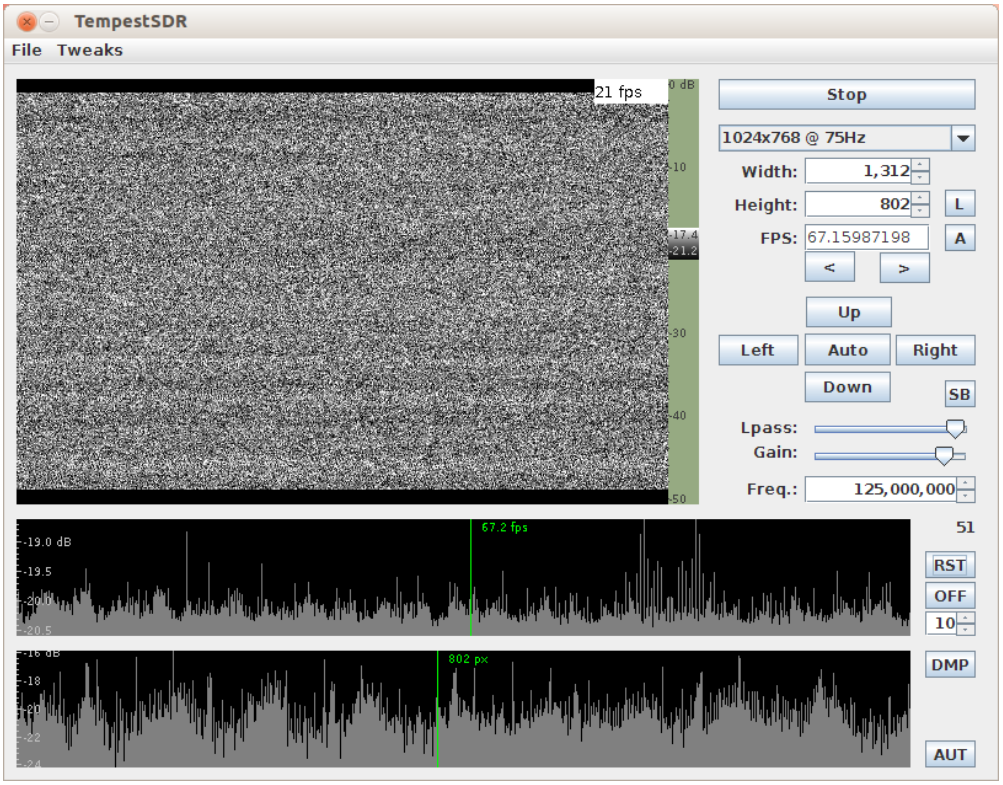


Figure 4.3: We can see two spikes in the top autocorrelation plot around 60 fps that is responsible for the target refresh rate.

on the same frequency may be more prominent so manual adjustment would be required. The autocorrelation plots are interactive and the user can zoom with the mouse wheel and pan around for choosing small details in the plots. Now we click on the 60.3 fps spike on the top plot and the 628 px spike on the bottom one.

If we turn on the "Lpass" averaging, we will get rid of most of the noise and improve the signal to noise ratio bringing up the weak video signal (Figure 4.4). We can clearly see the synchronisation pulses. We are also only seeing rapid changes in colours, so we know this is an analogue signal. Digital signals will also show activity in regions that have constant intensities. That's why digital signals could be eavesdropped easier – they emit more energy because of the sharp edges of the individual bits. However, we are apparently dealing with a VGA signal with resolution 800×600 @ 60. Note that the system

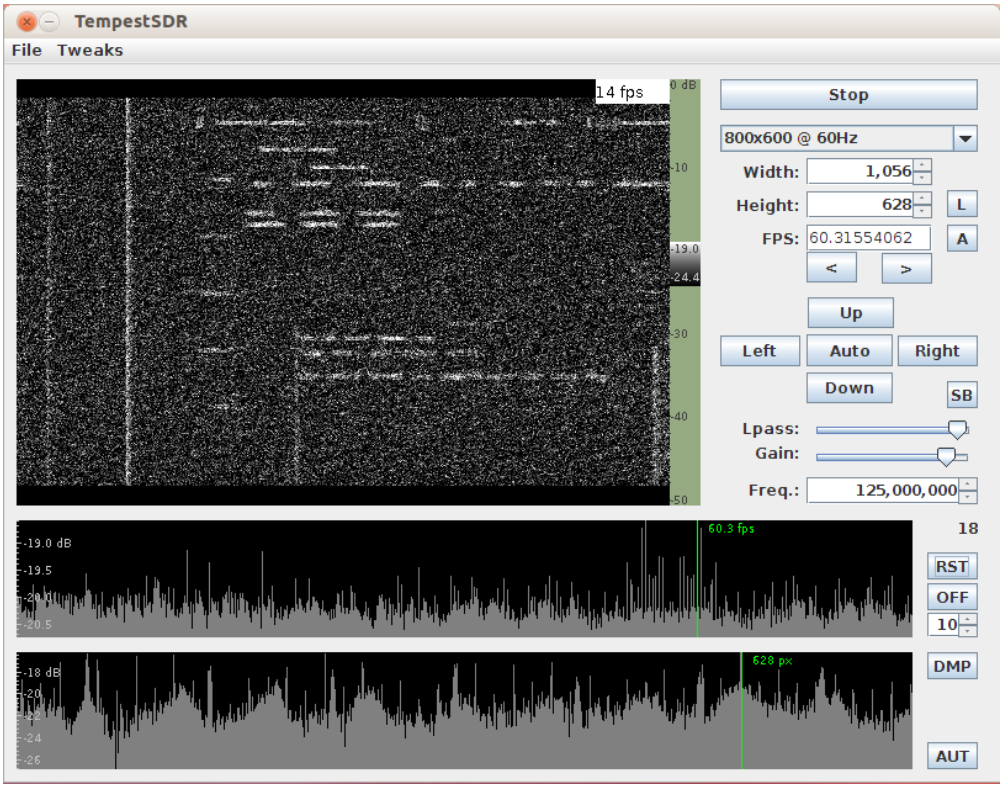


Figure 4.4: We can see two spikes in the top autocorrelation plot around 60 fps that is responsible for the target refresh rate.

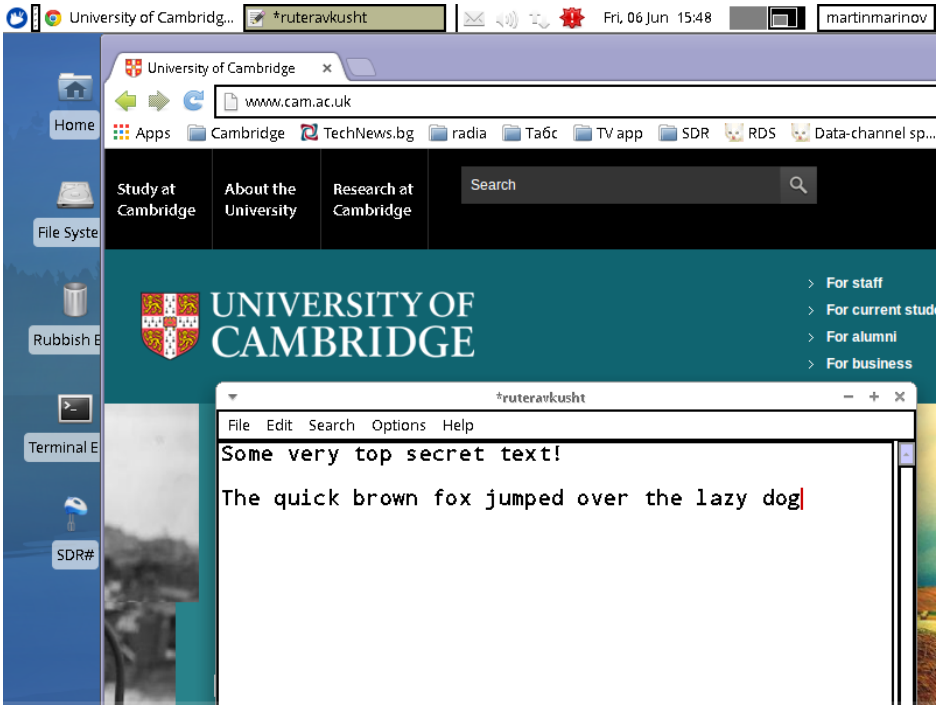
auto-detected it (it shows up in the least below the "Stop" button) just from our mouse clicks.

We see another autocorrelation peak at 61 Hz. This is aliasing due to the two strong synchronisation lines that appear in the video (you can see them on the left). By clicking on the other peak, we can indeed confirm our suspicion that it is just aliasing. Now that we have a signal, we can try to improve it. Since the signal is strongly polarized, changing the antenna angle and position can have a dramatic effect on the end result. Now we can increase the sampling rate of the USRP to improve the resolution of the obtained image.

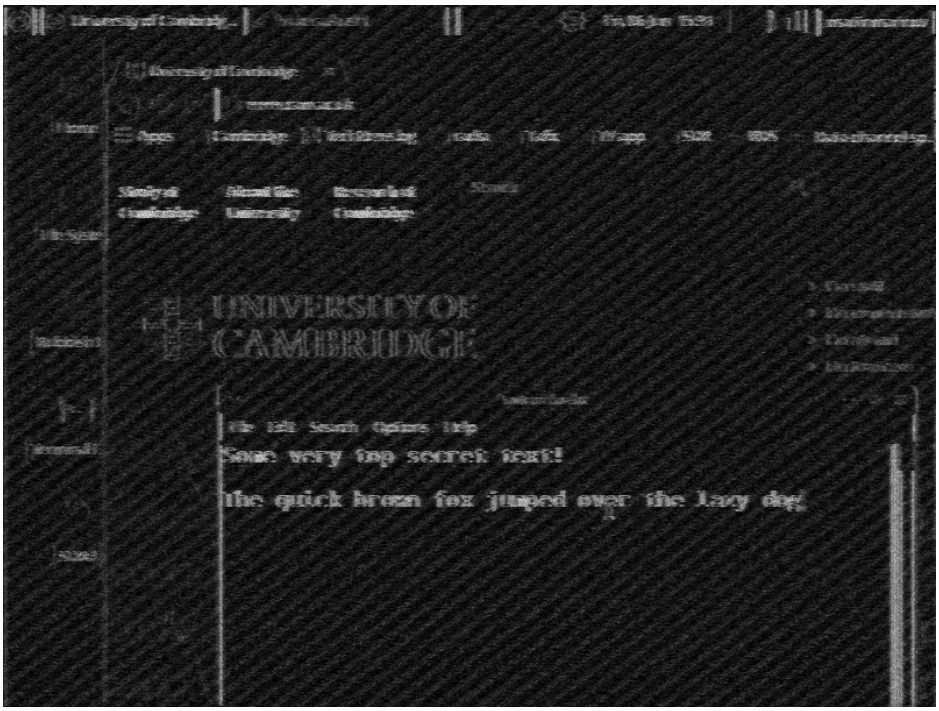
4.3 Conclusion

If we obtain a good enough signal, we can reconstruct text from the screen. Compare on Figure 4.5 the actual image shown on the target screen and the best estimate obtained from the hallway just outside the room with the target with 32 MHz sampling rate. We can in fact clearly read the text in the text editor.

This shows that a practical attack is possible even from an adversary on a low budget with very little knowledge of the DSP. We also demonstrated that we can deduce the video parameters of a screen remotely in a real-world environment. A more sophisticated attacker, could use better equipment with a lower noise floor and higher antenna gain to undertake such an attack from an even further distance. This should open the discussion about what could be done to prevent compromising emanations leaking from cables.



(a) Screenshot of the target monitor.



(b) Best result (blanking cropped out).

Figure 4.5

Chapter 5

Implementation

5.1 Hardware

Figure 5.1: From left to right: MSi3101, AverMedia antenna and USRP B200

There is a wide range of off-the-shelf Shardware front-ends that could be used with the system. An attacker can choose one depending on their re-

quirements. A quick overview of some of the supported available hardware is outlined below.

5.1.1 Ettus Research USRP

Maximum sampling rate:	56 Mhz (B200)
Frequency Range:	70 MHz - 6 GHz
Dimensions:	97 mm × 155 mm × 15 mm (B200)
OS support:	All platforms (Windows support via ExtIO)
Dropped samples detection:	Yes (Except on Windows)

The maximum sampling rate varies between the different models as well as the dimensions. I have used the USRP B200 throughout this report.

Pros:

- A variety of models available in multiple price ranges.
- High enough sampling rates for most eavesdropping scenarios.

Cons:

- Some models need external power supply. B200 in particular does not.
- May need some analogue preconditioning for good eavesdropping results.

5.1.2 Mirics FlexiTV^{TM}MSi3101

Maximum sampling rate:	8 Mhz
Frequency Range:	64 MHz - 240 MHz and 470 MHz - 960 MHz
Dimensions:	25 mm × 80 mm × 10 mm
OS support:	Windows only
Dropped samples detection:	Yes

It is the best low end solution but it won't achieve the receiving distance or the quality that one would get with a higher end device.

Pros:

- Portable and unsuspicious looking – just a typical USB dongle.
- Inexpensive – under £100.

Cons:

- Low sampling rate and low sensitivity. Good eavesdropping results would be obtained if the target screen resolution is low enough so that κ (refer to (3.15) for definition) is reasonable and the signal is strong enough.
- Tuning gap between 240 MHz and 470 Mhz. If the emanations fall within this range, the Mirics dongle will fail to pick them up.

5.1.3 Windows ExtIO

The plug-in supports variety of devices. It is a standard plug-in that is used in a number of software defined radio solutions. However, it does not take into account for dropped samples. This means that if sample loss did occur, this would be very visible on the screen rendering eavesdropping practically difficult. It also only runs on Windows.

5.1.4 Antennas and Preconditioning

For the experiments in this report, I have used a small dual-pole antenna by AverMedia designed for digital TV reception. This showed to be enough for eavesdropping from an adjacent room to the one containing the target device. However, for impressive demos that could potentially operate from several tens of meters, Kuhn recommends some additional analogue preconditioning.

Kuhn experimented with a tunable band pass filter and a 30 dB antenna amplifier before the B200 to improve the noise level. He comments that the setup delivers comparable quality to the highly specialised Dynamic Sciences

R1250 receiver that he used in his experiments [6]. He also used a professional log-periodic antenna for improved reception.

5.2 Architecture

The whole system consists of two parts - a library (called **TSDRLibrary**) and a host program. The library is the main deliverable of the project. It is responsible for decoding the emanated signal – from obtaining the data from the hardware to processing it. However it can't run as a stand-alone application. It needs a host program to control it via an API (defined in *TSDRLibrary.h*) and to receive the decoded video frames. The host program could be written in any programming language that can use native libraries.

However, the package also comes with a pre-built Java based host program. The provided implementation allows for full manual control over the library via a graphical user interface (GUI). It is very useful for real-time monitoring and interacting asynchronously with the library while the processing is taking place. In practice it is a single executable that has TSDRLibrary built in. This is what the report has been referring to as "the system" so far – the Java GUI combined with TSDRLibrary. An end user would never notice that there is another layer underneath the GUI. All of the examples and screen-shots given here have been obtained via the provided Java based GUI while it is controlling TSDRLibrary.

5.2.1 The Library

The library relies on plugins (called **TSDRPlugin**) to provide a stream of IQ samples and means to control the Shardware in real time. The data sources are shared libraries that are dynamically loaded at runtime by the TSDRLirbary.

The host program's job is extremely simplified. In a nutshell, it sets up the library and runs it. Every time there is a new video frame available, a

callback would be invoked via TSDRLibrary so that the host program could process the frame (display it, save it, etc). An expected sequence of API calls to get video data would be:

1. Dynamically load the library (if it is not statically compiled with the host program) using the methods provided by the specific operating system. This will give the host program access to the TSDRLibrary APIs.

2. Allocate an instance of the processing pipeline with *tsdr_init*.

3. Set target display resolution and frame rate with *tsdr_setresolution*. This could be called interactively while the processing is running to dynamically change the parameters at runtime.

4. Load a pre-compiled plug-in by invoking *tsdr_loadplugin* specifying the location of the shared library that represents the plugin and any parameters that the user wants to pass to the plug-in. This plugin will talk to the hardware and provide a steam of IQ data to the library for processing.

5. Start the processing by running *tsdr_readasync*. The current thread would be used to start polling the hardware so it will block until processing has finished.

6. The *tsdr_init* and *tsdr_readasync* allow for callback functions to be registered with TSDRLibrary. The callbacks are just C functions that reside in the host program that will be called as soon as a new video frames, autocorrelation data or other messages are available. Processing data within those callbacks is not thread safe.

7. Call *tsdr_stop* to stop the processing and unblock the thread that originally called *tsdr_readasync*. Soon after calling *tsdr_stop*, the callbacks will stop receiving data.

If the plug-in supports it, *tsdr_setbasefreq* can be used to set the base frequency of the hardware device. A few other API commands allow the host program to do manual position adjustment of the video frame, control the

amount of lowpass applied and set the gain of the hardware device (if supported). There are some advanced features that could be turned on using the *tsdr_setparameter* API call which include the automatic frame borders detection, automatic frame rate synchronisation and a few other tweaks to the digital signal processing pipeline. It is up to the host program to use the autocorrelation to determine resolution and framerate which gives enough room for flexibility.

Each API call can return an error code in case it fails. The host application can also read a verbose version of the error message using the *tsdr_getlasterrortext* API call. This allows for intuitive and simple error handling.

5.2.2 Data Flow

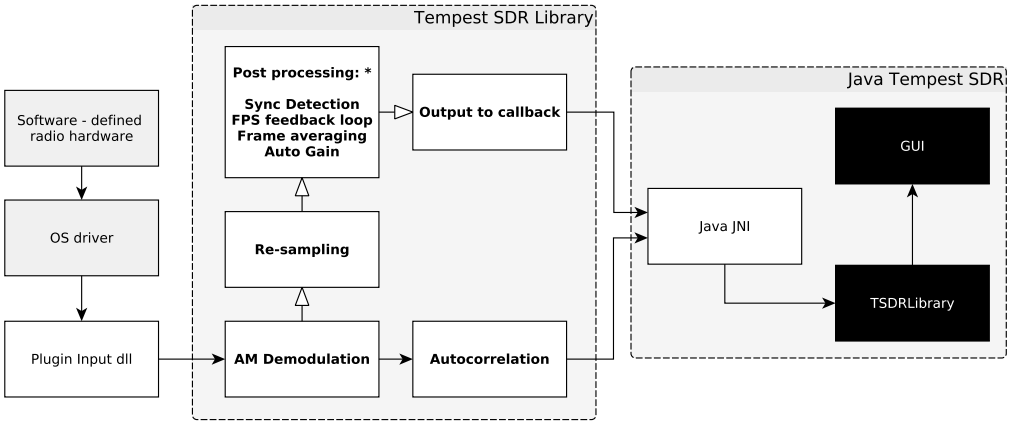


Figure 5.2: Data processing in the system using the Java GUI. Bold text represents different native threads. White background means code is written in C. Black background means code is written in Java. Gray background is for operating system related processes and external hardware. Arrows with white ends represent points where data could be dropped due to performance reasons. Arrows with black ends represent transferring full data that was processed in the previous step. The star on the Post processing stage means that the outlined sub-stages could be executed in different order depending on user preferences.

Figure 5.2 outlines the data flow and data processing paths within a typical set-up of the system[1]. It goes through the following steps:

1. RF data is converted into IQ samples in the Sdevice. These samples are sent to the PC for processing.

2. The hardware driver in the OS receives the samples and possibly queues them.

3. A TSDRPlugin either continuously polls the driver or receives a callback from it as soon as a block of data is available. It obtains the block of data and sends it to the TSDRLibrary via a callback.

4. The IQ data is inside TSDRLibrary. AM demodulation is done on the data. This runs in a separate thread so that the TSDRPlugin will have the chance to receive new data from the device as quickly as possible. The demodulated data is provided to the Autocorrelation thread that will calculate the autocorrelation of the incoming signal and notify the host program. If TSDRPlugin reports any dropped samples between calls, the autocorrelation is reset (we only want continuous data accumulated for autocorrelation).

5. The resampling logic runs in a separate thread. It matches the rate of the data from the sampling rate of the device f_s to the expected rate of $x_\mathrm{t} \cdot y_\mathrm{t} \cdot f_\mathrm{v}$.

6. Here some digital signal processing is done in a separate thread to enhance the signal. The order with which the operations are applied depends on the settings that a user will supply. By default the synchronisation pulses of a video frame are detected first. If the user has selected auto tracking, the current position of the horizontal blanking interval is compared with the last one. This gives a rough estimate on whether f_{fps} is off by a couple of parts per million from the real value. The process allows for keeping the picture steady on the screen. Then

[1] If instead the data is read from a file or network, we can regard the "Software - defined radio hardware" as the hard drive or the network layer.

lowpass in the time domain is done (a.k.a. motion blur) to de-noise a weaker signal using averaging. Finally, an autogain algorithm makes sure that the full dynamic range of the signal is equally utilised for the final image.

7. Here the host application receives a video frame via a callback. It is up to the host application to determine what to do with it. It can take as much time as it wants since this runs in a separate thread.

8. The signal leaves the library and goes into the host application. In this example this is our Java GUI. A Java Native Interface (JNI) code marshals the incoming data to a Java friendly format which is then passed to the Java Virtual Machine and the particular class instance that has invoked the whole processing.

9. Once the data arrives at the TSDRLibrary class instance, the main GUI will receive a notification via a callback and will dispatch the bitmap to be drawn on the screen.

The biggest problem faced was synchronising dropped samples. To illustrate the issue, imagine we are receiving samples at a rate of 50 mega samples per second. If each sample contains two floating point values (the I and the Q component), each of which being 4 bytes of data, then the total data rate coming from the hardware would be $8 \times 50 = 400$ megabytes per second! This is a huge amount of data. Since the configuration is running on a non-real-time operating system, the CPU might not be able to cope with such a fast stream. Furthermore there could be some hardware limits enforced as well, for example maximum throughput of the bus that is being used to transmit the data from the Sdevice to the PC.

This means that we will have a lot of dropped samples coming from the device. At the same time, even if we don't have any dropped samples, the CPU can still fail to process all of the desired data in real-time. This means that, in order to have the system run interactively, some samples that can't be processed in real-time should be dropped. This would allow for fresh data to be collected for processing. However having even a couple of dropped

samples is a problem. This is because video is continuous. Dropping a few pixels means that the whole frame synchronization could be broken.

Therefore the library employs an algorithm that allows it to drop samples only an integer multiple of $x_\mathrm{t} \times h$ at a time. This means that even if a single sample was dropped before processing, the algorithm will enforce a whole batch of $x_\mathrm{t} \times h$ samples to be dropped instead. This means that when the next frame arrives, it will start processing from where the previous frame had dropped samples. Therefore the picture will look seamless to the end user since the dropped samples will actually result in a lower frame-rate in the host application. This approach allows the system to run even on not so powerful configurations and still produce some output. Such synchronization points are denoted as white arrows on Figure 5.2.

5.3 Digital Signal Processing

Re-sampling

Digital re-sampling is used as the processing step that synchronises the incoming sample rate $f_\mathrm{s} = t_\mathrm{s}^{-1}$ to the desired pixel rate $f_\mathrm{p} = t_\mathrm{p}^{-1}$. This means that if each incoming sample contains information about a fraction of a pixel, after the re-sampling, each output sample will contain the combined information from the correct number of incoming samples so that it only contains the information about a single pixel. Alternatively if several pixels are contained in an incoming sample, the re-sampling will try to extract them into the correct number of output samples so that each of the output samples represents a single pixel intensity.

Mathematically, it sees a stream of samples coming at a rate f_s, interpolating their rate with f_p and then decimating the resulting stream of sampling rate $f_\mathrm{s} \cdot f_\mathrm{p}$ by a factor of f_s. This results in an output stream of rate $f_\mathrm{p} = t_\mathrm{p}^{-1}$. Each sample then will relate to a pixel value at that time. In mathematical

terms,

$$v[n] = x\left[f_{\mathrm{s}} \cdot \frac{n}{f_{\mathrm{p}}}\right],$$

where ($n \in \mathbb{Z}$) and $v[n]$ are the output pixel intensities and

$$x[n] = \sqrt{I_n \cdot I_n + Q_n \cdot Q_n}$$

are the incoming sample intensities. If this algorithm is implemented straight away, it would be called **nearest neighbour** re-sampling. Unless $x[n]$ is band limited to $\min(f_{\mathrm{p}}, f_{\mathrm{s}})$, $v[n]$ [2] will contain aliasing artefacts. Therefore, as a pre-processing step, we can apply a low pass filter to $x[n]$ with a cut-off frequency $\min(f_{\mathrm{p}}, f_{\mathrm{s}})$ as an anti-aliasing filter.

In practice this is an expensive operation since it involves an additional digital filtering step. Luckily, there is a good way of approximating a low pass filter while doing the re-sampling. It uses linear interpolation between two consecutive samples instead of the perfect sinc interpolation. In practice sinc interpolation is not achievable anyway since sinc is an infinite function in the time domain. The linear interpolation reduces the aliasing components. The actual implemented re-sampling algorithm in the project utilises the Brasenham's line algorithm[15].

There is another trick that is done to improve the anti-aliasing quality and possibly performance. Since the x_{t} of an image is only used to maintain the aspect ratio, the library assumes that $x_t = \lceil \frac{f_{\mathrm{s}}}{f_{\mathrm{v}} \cdot y_{\mathrm{t}}} \rceil$. This allows the host program to use whatever algorithm they desire to re-sample the output image to any resolution. Since the host program can use GUI acceleration for the re-sampling process, it can afford more expensive algorithms that result in less aliasing artefacts. Furthermore, when $f_{\mathrm{p}} > f_{\mathrm{s}}$ as in most real-world eavesdropping scenarios, the CPU will have to handle less data resulting in improved performance.

[2] Note that the notation $v[n]$ is equivalent to previously used v_n.

Auto Gain

The auto gain step ensures that the image uses the full dynamic output range. For example, if demodulated pixels $v[n]$ have the property $v_{\min} \leq v[n] \leq v_{\max}$ then we can construct a new stream of pixels $v'[n]$ such that

$$v'[n] = \frac{v[n] - v_{\min}}{v_{\max} - v_{\min}}$$

which now means that $0 \leq v'[n] \leq 1$.

Frame Averaging

The time averaging applies an infinite impulse response (IIR) low pass filter to given a stream of input pixels $v[n]$ to produce a stream of output pixels $v'[n]$ such that

$$v'[n] = \alpha \cdot v[n] + (1 - \alpha) \cdot v'[n].$$

This does inter-frame averaging sometimes known as motion blur. It removes random noise between consecutive frames, allowing only constant signal to come through. The weight of the filter could be adjusted by changing α from 0 to 1.

5.3.1 Synchronization Detection

In order to detect the blanking intervals in a frame, a few assumptions need to be done. What is characteristic about the intervals is that they do not contain any activity compared to the video region. This means that usually they are seen as a band of solid colour (see Figure 5.3). They either contain more energy than the video region or contain less energy. Therefore we can assume that the average energy of the blanking interval is very different from the average energy in the video region.

In order to detect the intervals, for performance reasons, we can decouple them into two 1 dimensional strips. We produce two strips – a vertical one,

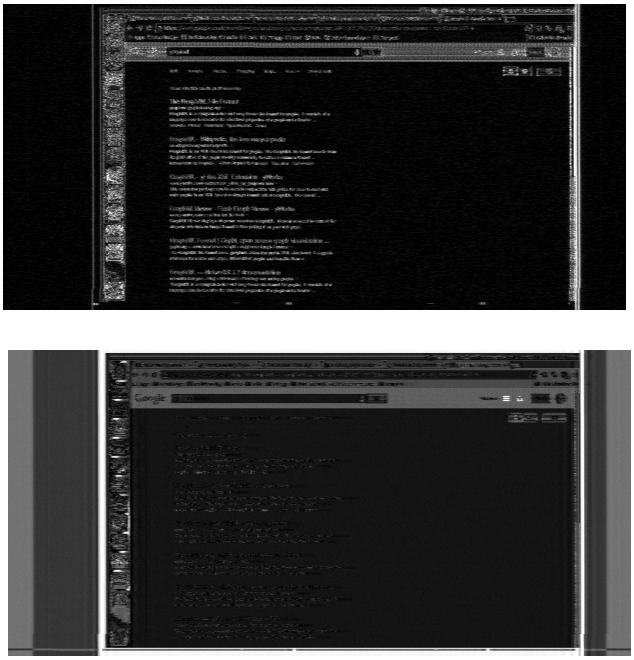


Figure 5.3: Compare the blanking interval of the same digital signal, obtained at two different frequencies of the spectrum. One of hem looks dark while the other is lighter and has some bands. The similarity is that they are very smooth and do not contain high frequency intensity changes.

containing the average of all of the lines and a horizontal one, containing the average of all of the columns in an image. Then we can do a circular low pass filter in order to get rid of any high frequency noise (such as text lines). Figure 5.3 shows how the averaged vertical and horizontal bands are derived and how they look after the lowpass filter.

Now we can use the assumed property that the synch bands have the most different average energy compared to video regions. Let's say a band has n elements $b[n]$. We now need to partition the two bands into two regions with width $2w$ and $n-2w$, centred at c and $n-c$. Then the value of the average per pixel difference would be:

$$\beta(w,c) = \left[\sum_{k=c-w}^{c+w} \frac{b[k \mod n]}{2w} - \sum_{k=2w-c}^{2(n-w)-c} \frac{b[k \mod n]}{2n-4w} \right]^2$$

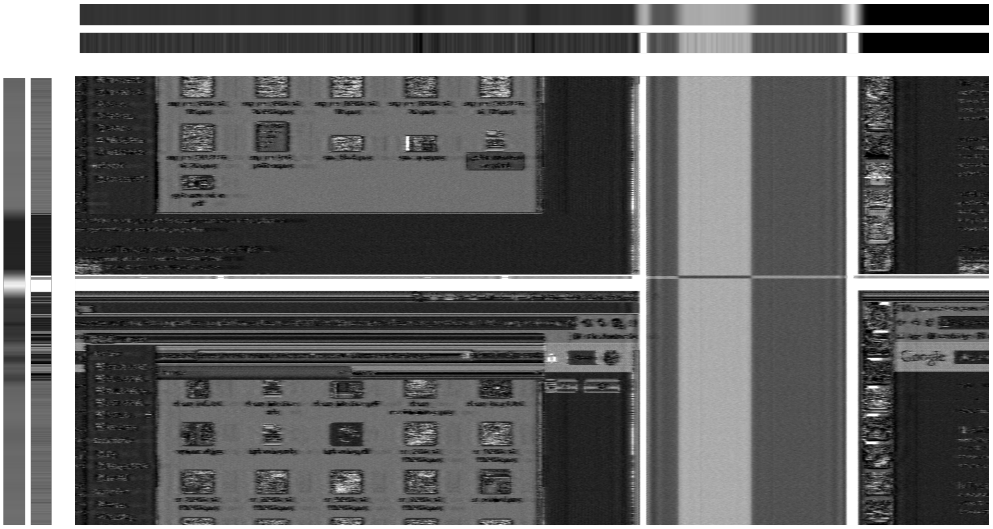


Figure 5.4: In order to detect blanking regions, first an average of the vertical and the horizontal pixels is taken. Then this average is blurred using a low pass filter with a Gaussian kernel.

We need to choose $2w$ and c so to maximise β. Since the constraint is $0 \leq 2w < n$ and $0 \leq c < n$, we can in fact calculate β for each possible input and find the parameters that maximise it. This is an algorithm complexity of $O(n^2)$ which explains why we decided to do that on two 1D arrays rather than on the whole 2D image.

However that algorithm won't always do the job. If the video signal has a strong horizontal or vertical component with a very short width or height, the algorithm will assume that this instead is the region of maximum difference since this will actually maximise β. We therefore need some realistic constraints about how small a blanking region can become. By observing a several video emanations, I came up with some arbitrary numbers of 5% of the width for horizontal blanking and 1% of the height for vertical blanking. These number can be adjusted (in the source code) depending on the judgement of the eavesdropper, but so far they have shown to perform well.

We can't easily change the asymptotical growth of the algorithm. However, we can reduce its execution time by putting some additional constraints. If we assume that the blanking region is always smaller than the video region, if

we assume that the blanking region always has a width of $2w$, then we know that $2w < n - 2w$, therefore we have an upper bound for the value of $2w < \frac{n}{2}$. Having the lower bound for $2w$ as well means that now the algorithm can run in real time for both strips. Figure 5.5 shows how such a strip ends up being partitioned by the algorithm which determines c - the position of the synchronisation region in a frame. This allows us to use this information in order to center the frame in the video window. The same approach is used for the vertical detection.

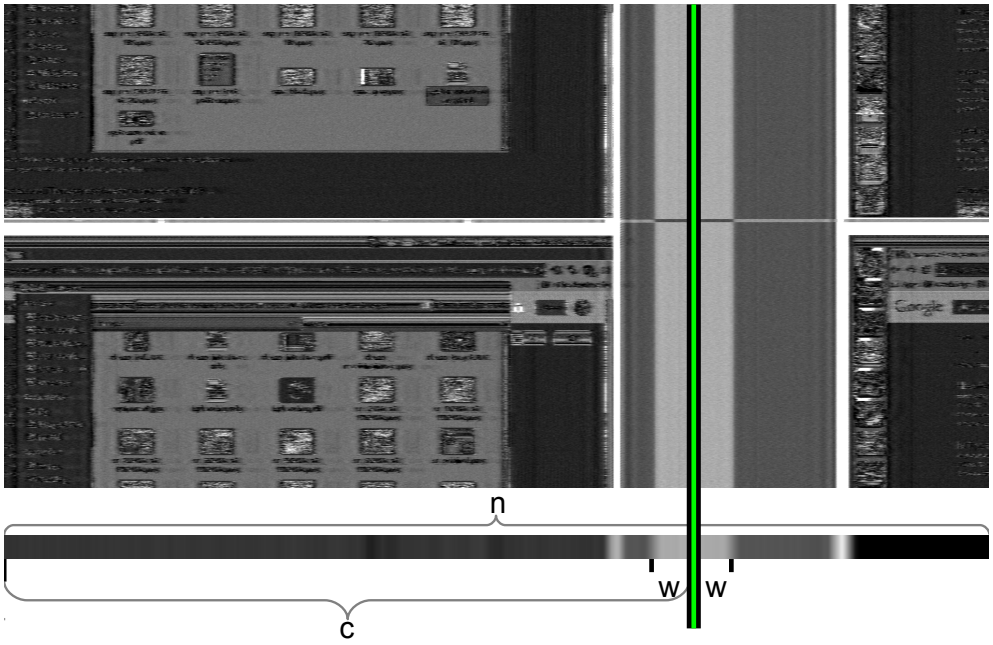


Figure 5.5: Visualising the relationship between c, w and n. The figure shows the value that give the maximum value of β for the frame.

5.3.2 Tracking the Frame Rate

5.3.3 Autocorrelation

5.3.4 Main Library

5.3.5 JavaGUI

Multiplatform

Graphical Interface

Visualisation

Chapter 6

Summary and Conclusions

[Polling machines]

6.0.6 Encountered Issues

[Summarize results]

[potential future work]

Bibliography

[1] Wim Van Eck. Electromagnetic radiation from video display units: an eavesdropping risk? *Computers & Security*, 4(4):269–286, 1985.

[2] Karen Jelved, Andrew D Jackson, and Ole Knudsen. Selected scientific works of hans christian ørsted, 1998.

[3] RFH Nalder. History of the royal corps of signals. *Royal Signals Institution, London*, 1958.

[4] Ross Anderson. *Security engineering*. John Wiley & Sons, 2008.

[5] US National Security Agency. Tempest: A signal problem. 1972.

[6] Markus G Kuhn. Compromising emanations: eavesdropping risks of computer displays. *University of Cambridge Computer Laboratory, Technical Report, UCAM-CL-TR-577*, 2003.

[7] Markus G Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In *Privacy Enhancing Technologies*, pages 88–107. Springer, 2005.

[8] James Clerk Maxwell. *A treatise on electricity and magnetism*, volume 1. Clarendon press, 1881.

[9] William L Briggs et al. *The DFT: An Owners' Manual for the Discrete Fourier Transform*. Siam, 1995.

[10] Johan Kirkhorn. Introduction to iq-demodulation of rf-data. *EchoMAT User Manual (FA292640)*, 15:4–10, 1999.

[11] Monitor Timing Specifications. Version 1.0. *Revision 0.8, Video Electronics Standards Association (VESA), San Jose, California*, 1998.

[12] Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[13] G Arfken. Convolution theorem. *Mathematical Methods for Physicists*, pages 810–814.

[14] Ronald Bracewell. The autocorrelation function. *The Fourier transform and its applications*, pages 40–45, 1965.

[15] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.