gr-tempest: an open-source GNU Radio implementation of TEMPEST

Federico Larroca, Pablo Bertrand, Felipe Carrau, and Victoria Severi
Universidad de la República, Uruguay. Email: {flarroca,pablo.bertrand,felipe.carrau,victoria.severi}@fing.edu.uy

***Abstract*—Like all time-varying voltage and current, a video interface connecting a PC to its monitor emits electromagnetic waves. The attack commonly known as TEMPEST (or Van Eck Phreaking) consists in receiving this signal and inferring the image being displayed on the monitor; that is to say, pointing an antenna to a PC and spying the monitor. This is a particularly interesting application for Software Defined Radio (SDR), as it requires modeling the signal and implementing a custom receiver. However, and although the first public demonstrations date back to the mid-80s by Wim Van Eck, no open-source implementation was available until Martin Marinov's TempestSDR was published in 2014. TempestSDR consists of a module written in C that takes care of the signal processing, plug-ins for various models of SDR hardware, and a Java-based GUI. This results in a multi-platform software that, although functional, it is difficult to extend or tweak. For instance, new plug-ins have to be written for new SDR hardware, or including filters or other DSP blocks in the signal's flow is not straightforward at all.**

To remedy this we developed `gr-tempest`, an open-source GNU Radio-based implementation of TEMPEST (available at https://github.com/git-artes/gr-tempest). This is an on-going project whose objective is to enable simpler experimentation by taking advantage of GNU Radio's functionalities and support. We describe the mathematical principles behind the TEMPEST attack and present how `gr-tempest` works. Furthermore, we show several real-world examples including both VGA and HDMI, and the fundamental differences between both types of signals. Finally, some of the advantages of using GNU Radio's framework are showcased by introducing modifications to the DSP chain that allows significant improvements of the resulting image with respect to the original method used in TempestSDR.

***Index Terms*—side-channel attack, eavesdropping attack, compromising emanations, software defined radio**

I. Introduction

TEMPEST is the name given by the United States' National Security Agency to the problem of involuntary electromagnetic emanations that could be compromising in terms of security [1]. It has a long history going back to the second World War, when Bell Lab's researchers found that, when being operated, their 131-B2 encryption machine produced spikes on a distant oscilloscope. A careful examination of these spikes could reveal the hidden message.

The first unclassified technical report regarding TEMPEST was published several years later by Wim van Eck [2] in 1985 and it focused on Cathode Ray Tube (CRT) monitors, their cables, and eavesdropping the video being displayed. As a consequence, the term *Van Eck Phreaking* is sometimes used to refer to TEMPEST in the context of video eavesdropping from compromising emanations. In 2003 Markus Kuhn publishes a report on how to spy (and protect) modern video displays [3]. The spying system incorporated FPGA boards and an AM receiver, obtaining better results but still requiring high-cost equipment with specific and fixed characteristics.

Around the same time, Software-Defined Radio started to gain traction. In a nutshell, the idea is to implement in software as much as possible of a radio communication system, most of which were traditionally implemented in hardware. Typically it consists of a generic hardware that basically moves the signal from pass-band to base-band, performs sampling and feeds these samples to an ordinary PC, which in turn processes them in order to receive the message. Transmission is also possible depending on the hardware, in which case samples are fed by the PC to the generic hardware. This paradigm is increasingly extending not only as the preferred research and development method in industry, but due to the availability of low-cost hardware it is also used for teaching [4], [5].

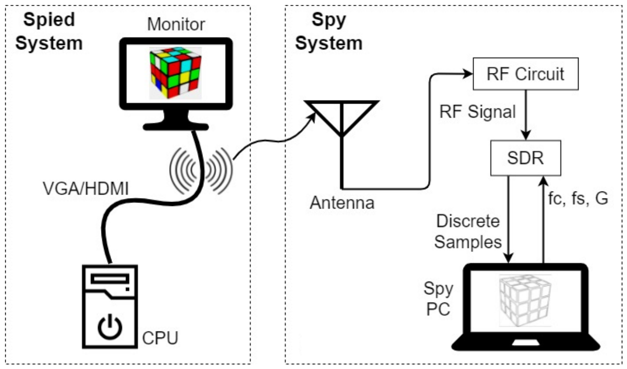


Fig. 1: SDR-based TEMPEST implementation. The spying PC is equipped with an SDR hardware from which it receives discrete samples and may in turn configure some parameters (f_c, f_s and G stand for carrier frequency, sampling frequency and gain respectively). The RF Circuit may include further amplifiers and/or filters.

The problem of TEMPEST, which requires a custom receiver and much experimentation, is a clear example of an application where SDR acts as a facilitator (see Fig. 1 for the basic setup). In this context, Martin Marinov published TempestSDR in 2014 [6], the first open-source implementation of TEMPEST. It consists of a collection of plugins that interface with the SDR hardware, some modules that perform the signal processing, and a GUI from which the system may be controlled and the spied image is displayed.

The general purpose hardware resulted in an affordable system, specially when compared with the equipment available in the market [7]. However, given its self-contained nature, TempestSDR is not straightforward to modify or update. For instance, new modules have to be written as new SDR hardware is released (or drivers are updated). Arguably more importantly, it is not easy to experiment with modifications or extensions to the signal processing chain.

In order to address the above issues, we developed and present here `gr-tempest` (code available at https://github.com/git-artes/gr-tempest). Instead of coding the complete system, we rely on GNU Radio [8], an extremely popular framework for SDR software. GNU Radio basically provides a framework to interconnect the blocks that compose the receiver. Furthermore, several signal processing blocks (ranging from simple mathematical operations to filtering) are already included, as well as blocks to interface with virtually all SDR hardware. Finally, new blocks are relatively easy to implement: the developer has to focus on implementing a few methods, and GNU

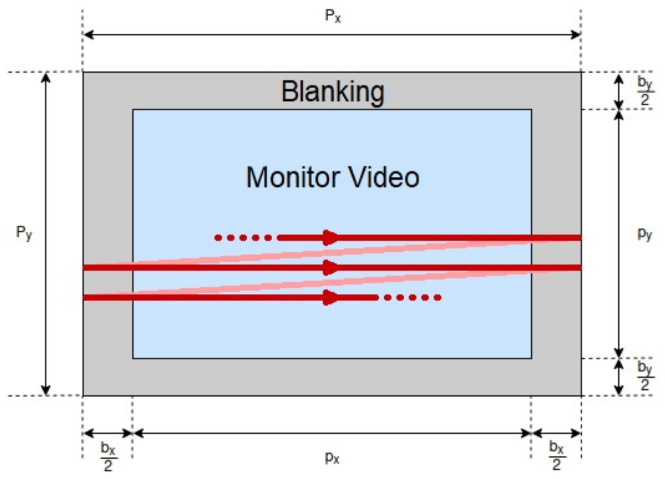


Fig. 2: The signal as generated in the video cable and the parameters involved. In red, the order by which pixels are sent over the cable. During the blanking intervals zero-valued pixels are sent.

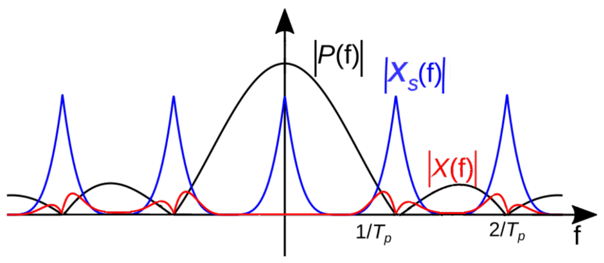


Fig. 3: The spectrum of the signal on the cable ($X(f)$, in red) results from the multiplication of the Fourier Transforms of the pulse ($P(f)$, in black) and the pixel sequence ($X_s(f)$, in blue).

Radio takes care of, for instance, executing all blocks involved in the receiver chain.

The rest of the article is structured as follows. The next section presents TEMPEST in more detail, describing the mathematical model of the resulting electromagnetic signal. In Sec. III we discuss the implications of this model on the SDR hardware and software requirements, and we further refine the signal model to consider the hardware involved. Then, in Sec. IV we present a first version of `gr-tempest`, which basically emulates TempestSDR and will help to illustrate some of its disadvantages. We showcase the advantages of using GNU Radio by extending the vanilla system in Sec. V. These modifications rely on the refined model we derive in this article and have a significant impact on the resulting image quality. The article is concluded in Sec. VI.

II. TEMPEST: Mathematical Principles

To fully understand TEMPEST we need to first describe the signal on the cables, which will in turn generate electromagnetic fields which are what we will ultimately spy on. For simplicity we will first focus on the analog case (i.e. VGA), and leave the digital (i.e. HDMI) case for the end of this section.

A. Analog Displays

In this case, the signal on each of the three color pins is basically a PAM (Pulse Amplitude Modulated) signal, with a Non-Return-to-Zero shape (i.e. a rectangular pulse). The height of each pulse corresponds to the intensity of the corresponding pixel (spanning between 0 and 0.7 V), which are generated sequentially starting from the top-left part of the image, and finishing at the bottom right. Furthermore, and mostly due to historic reasons, both a horizontal and vertical blanking intervals are included (see Fig. 2).

The figure also introduces the parameters used for the involved magnitudes. Since a full frame is formed by $P_x \times P_y$ pixels and are transmitted at a refresh rate of f_v, the duration of each pixel is given by the expression:

$$T_p = \frac{1}{P_x P_y f_v}. \tag{1}$$

The stream parameters are standardized by the VESA [9] to guarantee compatibility between systems and monitors. The resolutions are noted as $p_x \times p_y@f_v$ (with for instance $P_x = p_x + b_x$), so they are defined by the vertical and horizontal video size at a certain refresh rate. An example of resolution relevant to this work is $1920 \times 1080@60$ Hz. The rest of the parameters, such as blanking sizes b_x and b_y, are obtainable from the standard once the resolution is known.

We thus have that the voltage signal in each of the three color cables is:

$$x(t) = \sum_k x[k]p(t - kT_p), \tag{2}$$

where $x[k]$ is the intensity of pixel k and $p(t)$ a rectangular pulse with width T_p. The Fourier transform of (2) reveals that the spectrum of the signal is

$$X(f) = P(f)X_s(f), \tag{3}$$

where $P(f)$ is the Fourier transform of the pulse $p(t)$ and $X_s(f) = \sum_k x[k]e^{-j2\pi kfT_p}$. This corresponds to the Discrete-time Fourier Transform (DTFT) of the pixel sequence evaluated at $\omega = 2\pi kfT_p$ (thus with a period of $1/T_p$).

Figure 3 shows an illustrative example of (3). Note that $x[k]$ is a sequence that typically presents high auto-correlation $R_x[l]$ around lags $l = \pm 1$, $l = \pm P_x$ (since neighboring pixels, both horizontally and vertically tend to be similar), and $l = \pm P_x \times P_y$ (since pixels of successive frames also tend to be similar). This results in a spectrum $X_s(f)$ with the highest levels of energy centered at low frequency, as the one depicted in blue in Fig. 3 (a pattern that is repeated every $1/T_p$, since as we mentioned before $X_s(f)$ is periodic). In any case, this is the signal that has the information about the pixel values.

Furthermore, since the pulse shape is (approximately) rectangular, $P(f)$ will result in a $\text{sinc}(fT_p)$ shape, with a null every $1/T_p$ (see the black curve in Fig. 3). The actual spectrum of the signal in the cable (depicted in red in Fig. 3) is the multiplication of both $P(f)$ and $X_s(f)$. Note that its baseband components will not propagate (and are thus not drawn in Fig. 3). In order to receive the largest energy through an antenna we should tune our receiver to one of the harmonics at multiples of $1/T_p$. However, this is precisely where the nulls of $P(f)$ lie, resulting in a highly distorted signal.

Finally, and this is is also true for the digital display case, note that we will actually receive a combination of the signals corresponding to all three colors.

B. Digital Displays

The most important difference between VGA and HDMI is that each color now has an 8-bit depth, that is then encoded into a 10-bit word [10]. The encoding process is known as Transition-Minimized Differential Signalling (TMDS), and is divided into two stages. First, each bit of the original byte is either XOR or XNOR with the previous one (the first bit is left as-is), and the 9th bit indicates which operation

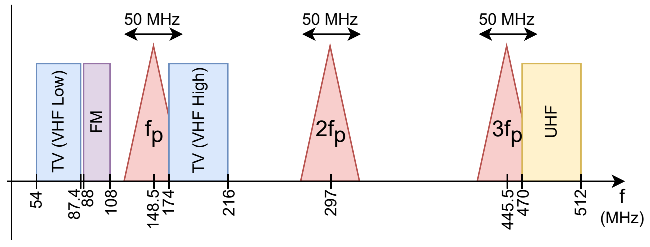


Fig. 4: The first three harmonics of $f_p = 1/T_p$ for the $1920 \times 1080@60Hz$ resolution, and the most important expected interferences. The second harmonic is the one used in our experiments.

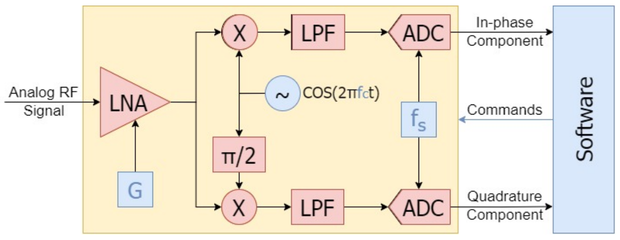


Fig. 5: Diagram of the SDR hardware. The parameters configurable from software are highlighted in blue. SDR software typically works with a complex value for each sample, where the in-phase and quadrature components correspond to the real and imaginary parts respectively.

took place. Secondly, the number of 0s and 1s in the stream are evened out by inverting (or not) the first eight bits in this word. The tenth bit indicates whether this inversion took place. Furthermore, during blanking two bits of control data are transmitted, where each of the four possibilities are represented by a 10-bit word.

The result is a signal $x(t)$

$$x(t) = \sum_k b[k]p(t - kT_b), \tag{4}$$

where T_b is the bit time and $b[k]$ is the voltage level representing the k-th bit (which may be either unipolar or polar signaling depending on the mode). That is to say, very similar to the VGA case (i.e. (2)), but with a 10 times shorter period (i.e. $T_b = T_p/10$) and $b[k]$ now referring to the bits that represent each pixel.

The resulting spectrum is relatively similar to the one obtained by VGA (cf. Fig. 3), although T_p is naturally now substituted by T_b. Furthermore, note that $X_s(f)$ will not have most of its energy at baseband due to the encoding (for instance, there is no DC if differential mode is used). However, correlation between pixels is still true, and we should thus expect an $X_s(f)$ with the largest components at multiples of $1/T_p$. All things considered, the resulting spectrum $X(f)$ should now have most of its energy at the harmonics of $1/T_b$ plus a multiple of $1/T_p$.

III. TEMPEST AND SOFTWARE DEFINED RADIO

The characterization of the signals we presented in the previous section has some important consequences regarding both hardware and software. For the sake of clarity, we will focus on the $1920 \times 1080@60Hz$ resolution (i.e. the popular 1080p video mode) in the VGA case.

A first aspect to consider is that at this resolution, the pixel rate $f_p = 1/T_p$ is roughly equal to 148.5 MHz. This means that strong interference should be expected at, for instance, the first harmonic (which should be the one with the strongest energy of $X(f)$). In our experiments we have thus used the second harmonic (see Fig. 4).

Secondly, only extremely high-end SDR hardware is capable of operating at a sampling rate that exceeds the 100 MSps. If budget is a constraint, then one may operate at 50 MSps for around 1000 USD or less (see for instance [11], [12]). This means that the system will actually receive a decimated version of $x(t)$ (i.e. roughly one sample out of every three pixels).

More in detail, let $h_{LPF}(t)$ be the impulse response of the low-pass filter used by the SDR before sampling the signal (see Fig. 5) and $H_{ch}(f)$ the transfer function of the base-band representation of the channel. For instance, if we are tuning our SDR to $f_c = 2/T_p$, and using the same notation as in Fig. 3, we would obtain $H_{ch}(f) = P(f - 2/T_p)$ [13, Ch. 9]. Naturally, other impairments may be incorporated to $H_{ch}(f)$ (such as attenuation or the transfer function of the VGA connectors and cable that act as an antenna), but in any case we may only assume an approximate (at best) knowledge of $H_{ch}(f)$.

Furthermore, let $T_s = 1/f_s$ be the sampling period used by the SDR, and $f_\triangle$ the inevitable frequency difference between the SDR and the actual harmonic of the pixel rate (i.e. in our case $f_c = 2/T_p + f_\triangle$). Thus, and ignoring possible interference and noise, we will obtain the following complex signal from the SDR hardware [14, Ch. 5]:

$$y[l] = \sum_k x[k]g(lT_s - kT_p)e^{-j2\pi f_\triangle lT_s}, \tag{5}$$

where $g(t) = h_{LPF}(t) * h_{ch}(t)$, $h_{ch}(t)$ is the inverse Fourier Transform of $H_{ch}(f)$ and $*$ stands for the convolution operation. It is important to highlight that $x[k]$ is a purely real sequence. However, complex values will be obtained from the SDR due to the presence of $h_{ch}(t)$ and the frequency error $f_\triangle$, which will constantly rotate the signal in the complex plane.

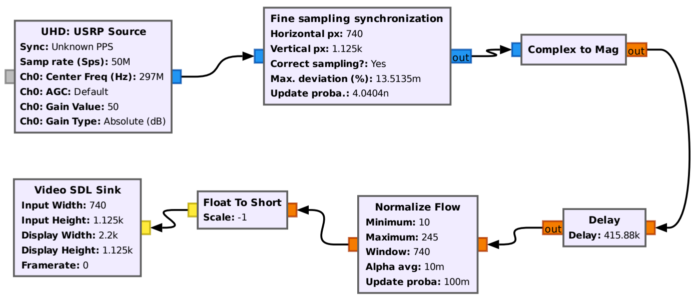
Finally, real-time operation at 50 MSps is a very challenging task. One may always envision an offline operation, where the signal is recorded and then processed. However, this is both impractical in terms of storage capacity (the size of the file with the IQ samples rapidly grows) as well as the attack *per se* (being able to operate the system online would be much more interesting). This calls for a system that is as lightweight as possible, although computing power is naturally another important constraint (i.e. a relatively powerful PC will be necessary to operate `gr-tempest`).

IV. VANILLA `GR-TEMPEST`

A. Implementation

Let us first discuss an implementation of `gr-tempest` that basically emulates TempestSDR. Figure 6 shows the corresponding flowgraph. This is the term used in GNU Radio to refer to the file specifying how blocks are interconnected. In this case, it is a simple succession of six blocks, which we briefly comment now.

1) `UHD: USRP Source`: The first block is naturally the SDR hardware. We have used a USRP B200-mini [11], and continuing with our ongoing example, we are going to use a sampling rate $f_s = 50$ MHz and tune the SDR to the second harmonic of the pixel rate corresponding to the $1920 \times 1080@60$Hz resolution (i.e. $f_c = 2 \times 148.5 = 297$ MHz). Note that using other SDR hardware is as simple as changing this block to the one corresponding to the available model. Virtually all SDR hardware is supported by GNU Radio. Finally, in all the results we show here we have also used a combination of low and high pass filters [15], [16] (to act as band


UHD: USRP Source
Sync: Unknown PPS
Samp rate (Sps): 50M
Ch0: Center Freq (Hz): 297M
Ch0: AGC: Default
Ch0: Gain Value: 50
Ch0: Gain Type: Absolute (dB)

Fine sampling synchronization
Horizontal px: 740
Vertical px: 1.125k
Correct sampling?: Yes
Max. deviation (%): 13.5135m
Update proba.: 4.0404n

Complex to Mag

Delay
Delay: 415.88k

Normalize Flow
Minimum: 10
Maximum: 245
Window: 740
Alpha avg: 10m
Update proba: 100m

Float To Short
Scale: -1

Video SDL Sink
Input Width: 740
Input Height: 1.125k
Display Width: 2.2k
Display Height: 1.125k
Framerate: 0

Fig. 6: The GNU Radio flowgraph corresponding to the vanilla `gr-tempest`.

pass RF filter, absent in the USRP B200-mini), an LNA [17] and a very simple whip antenna. Although relatively basic, the setup allows to spy at about 5 meters.

2) `Fine sampling synchronization`: The second block is arguably the most challenging one. Recall that we are not sampling at the pixel rate, but we still need exactly $H \times V$ samples for each frame worth of samples obtained from the SDR hardware (since they will ultimately be fed to a block that displays video at this resolution). The integer V has to correspond to the vertical resolution (in this case $P_y = 1125$; recall that the blanking periods increase the image's height and width), whereas H may actually be any integer. By default we use the closest integer to the horizontal resolution interpolated at f_s (i.e. $H = \text{round}(P_x \times f_s/f_p)$) so as to minimize the loss of information.

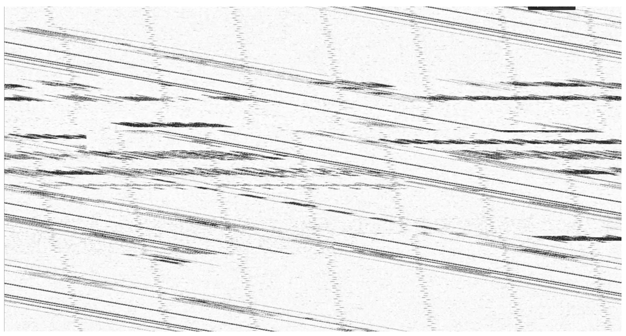
In order to produce these $H \times V$ samples, we will first compute the signal's autocorrelation around a lag corresponding to $\tau = 1/60$ seconds (i.e. the number of samples corresponding to a full frame). As we mentioned before, since frames tend to be very similar, there should be a large peak at that point of the auto-correlation. A rational resampler with an interpolation ratio equal to $H \times V$ divided by the actual position of the peak (in number of samples) will obtain the $H \times V$ samples we need. The estimated peak position is averaged over each iteration so as to filter-out noise and obtain a more stable interpolation ratio.

3) `Complex to Mag`: This block, part of the core of GNU Radio, simply computes the magnitude of the complex input, and it is what TempestSDR refers to as Amplitude Demodulation. In Marinov's thesis this was used without much analysis (see [6, Sec. 3.3.1]), probably following Markus Kuhn's previous work that used an AM receiver to downconvert the signal. Actually, its most important impact and utility lies in eliminating the complex exponent in (5) that would rotate the samples, and intuitively it acts as a measure of how much energy resides in that sample. However, as we will see in the next section, this non-linear transformation further distorts the signal, and much better images may be obtained by carefully considering the actual model we derived in (5).

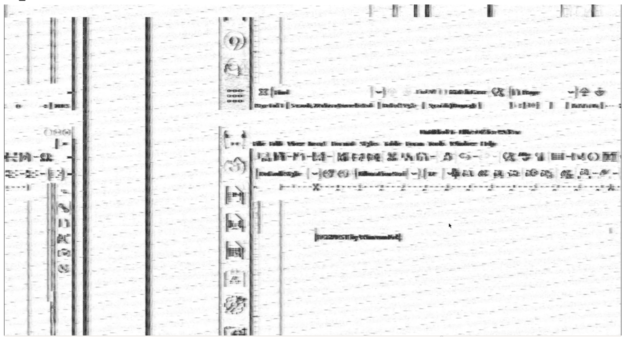
4) `Delay`: At this point we have a stream of real-valued samples corresponding to a sequence of $H \times V$ samples per frame. However, the beginning and end of each frame was not identified. This block (part of the core of GNU Radio) delays the signal a number of samples adjustable by the user. The objective is to manually center the image.

5) `Normalize flow`: This block simply prepares the stream to be displayed, by automatically adjusting its range.

6) `Float to Short`: In addition to changing the data-type for the next block, we may optionally negate the stream samples, which typically increases the visibility of the displayed image.



(a) Spied image when *not* using the `Fine sampling synchronization` block.



(b) Spied image when using the `Fine sampling synchronization` block. Note that the image is not centered, highlighting the role of the `Delay` block.

Fig. 7: An example of a spied image in the vanilla `gr-tempest` on a VGA interface. All images are better viewed in a monitor.

7) `Video SDL Sink`: This is a block (part of the core of GNU Radio) which takes a stream of shorts and displays it as a video. The resolution is adjustable and as we mentioned before we use $H \times V$, meaning it will take chunks of this many samples and display it as a frame (with a frame rate inferred from the incoming sample rate). Note that the displayed image is interpolated horizontally to restitute its original aspect ratio (through the `Display Width` parameter).

B. Experimental Results

An example result of this vanilla `gr-tempest` may be seen in Fig. 7 (the spied interface is VGA). The upper screenshot shows the displayed image when the `Fine sampling synchronization` block is *not* used. Naturally, we obtain a strongly tilted image, which added to its constant horizontal movement results in a useless system.

On the other hand, a correct interpolation results in a steady and relatively clear image, where for instance icons are clearly discernible. Furthermore, the spied image contains only the vertical borders of the original one. Recall from Fig. 3 that $H_{ch}(f)$ is roughly a high-pass filter, and that pixels are successively sent on the horizontal sense (cf. Fig. 2). This, together with the fact that we took the magnitude of the samples, results in a sort of vertical border detector.

V. First Improvements

The current version of `gr-tempest` includes several improvements to the vanilla implementation. Some are mostly focused on performance and visualization, as the ones we list below:

- Automatic vertical and horizontal centering of the spied image. That is to say, automatically setting the parameter of the `Delay`

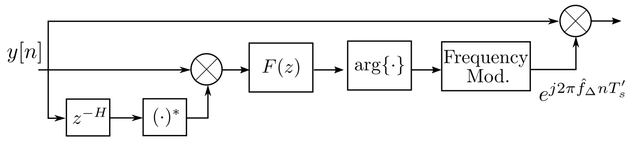


Fig. 8: A feed-forward frequency correction system that uses the fact that pixels a line apart (i.e. vertically contiguous) are very similar.

block so that the displayed image is centered. This is achieved by averaging both vertically and horizontally the image, where low values are indicative of the blanking periods. This is important when samples are dropped by the SDR hardware, since if a fixed delay is used, at that moment the image moves and its position has to be manually adjusted again.

- Hardware-based resampling. When using a large sampling rate, necessary to obtain a clearer image, the PC may struggle to process the signal, resulting in the dropped samples we mentioned before. One of the most computationally intensive tasks is to actually interpolate the signal. In addition to computing the resampling rate, this is the second task of the `Fine sampling synchronization` block. To alleviate the computational burden we may instead leave this task to the SDR hardware. Several SDR hardware models support arbitrary sampling rates, such as most USRPs. We have thus implemented a new block that computes the correct sampling rate and re-configures the `UHD: USRP Source` block accordingly. Actually, all SDR hardware with a block in GNU Radio that supports message-based configuration are supported by this feature.
- Frame dropper. For mid-range PCs, even if resampling is performed in hardware, 50 MSps may still be an excessive computational burden. We may lower this even further by dropping complete frames. That is to say, instead of sampling at lower rates (and thus obtain a lower-quality image), we may instead lower the frame rate of the image we display.

A. *Frequency correction*

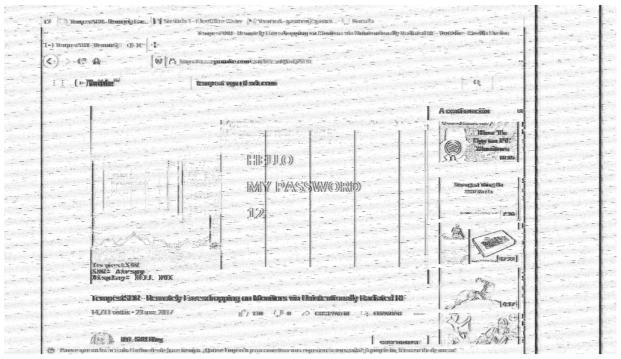
Let us now discuss with some detail some improvements that actually takes into account the model we derived in (5). As we mentioned before, in the VGA case there are two factors that complicates the visualization: the frequency error $f_\triangle$ and the blurring produced by $g(t)$. In TempestSDR the former was taken care of by using the magnitude of the complex sample, whereas the second was ignored.

However, the frequency error may actually be estimated by using again the fact that contiguous pixels (both horizontally and vertically) are typically very similar. Given this observation, and referring as $y[n]$ to the samples with the sampling rate corrected, a single-shot estimate of $f_\triangle$ is

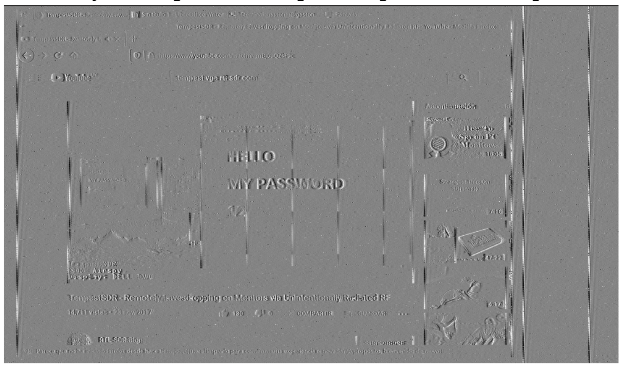
$$\hat{f}_\triangle = -\frac{1}{H2\pi T_s'} \arg\{y[n]y^*[n-H]\}. \tag{6}$$

This suggests a feed-forward frequency estimation system as the one depicted in Fig. 8, where the product $y[n]y^*[n-H]$ is filtered and then its argument is used by a digital frequency modulator to generate $e^{j2\pi \hat{f}_\triangle n T_s'}$, which is then multiplied by the original $y[n]$ to correct the frequency error (note that we have used T_s' to highlight that the original sampling rate T_s has already been corrected).

After correcting the frequency error, we may take the real part of the resulting complex, instead of its magnitude. As shown in Fig. 9, the effect on the spied image of doing this is appreciable in terms of the legibility of the text (where for instance the URL is now discernible).



(a) Spied image when using the magnitude of the signal.



(b) Spied image when using the real part of the signal after having corrected the frequency error by using the system in Fig. 8.

Fig. 9: An example of a spied image comparing the use of the magnitude of the signal, and its real part after being frequency-corrected. Note that the text is now clearly legible (e.g. the URL).

Note however that there is a small remaining frequency error. Indeed, (6) implicitly assumes that the phase different between samples corresponding to consecutive lines is less than 2π. In any case, this coarse frequency correction along with taking the real part of the signal clearly results in a much better image.

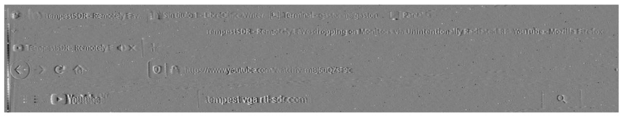
B. *Equalization*

The next natural step in processing (5) is to remove $h_{ch}(t)$ from $g(t) = h_{LPF}(t) * h_{ch}(t)$ in order to obtain a deblurred image. As we mentioned before, we have an approximate knowledge of $h_{ch}(t)$ at best. Even without considering other factors that may affect the signal, the pulse $p(t)$ used to shape the pixels' signal is only approximately rectangular (the VESA standard [9] specifies the pulse with generous tolerance).

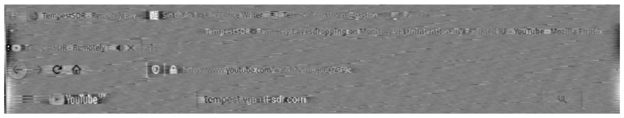
We will nevertheless evaluate the benefits that equalization may bring by using a simple zero-forcing equalization assuming a perfectly rectangular $p(t)$. Recall that $H_{ch}(f) = P(f - f_c)$, which has a null precisely at $f = 0$ (cf. Fig. 3). We will thus use a FIR filter $h_{eq}[n]$ such that its continuous time equivalent frequency response is equal to

$$H_{eq}(f) = \frac{1}{P(f - f_c) + \epsilon},$$

with ϵ a small constant. Computing the taps is relatively straightforward (we have used Numpy) and GNU Radio includes a FIR filter block where arbitrary taps may be used.



(a) The upper part of Fig. 9b, where we have corrected the frequency error and taken the real part of the signal



(b) The same part of the image, but we have additionally used a simple equalization filter.

Fig. 10: An example of the spied image when using (or not) the equalization, corresponding to the upper portion of the one shown in Fig. 9. Note how the text is even clearer, and certain images are now "filled".

Figure 10 shows the new results and compares it with the ones obtained when equalization is not used (i.e. Fig. 9). Note how text is even clearer than before. Furthermore, some of the images now have "fill". For instance, the YouTube logo or the padlock on the address bar, or even the Home or Reload buttons. However, interference is also highlighted by this somewhat simplistic filter. In any case, this illustrates both that there is much room for improvement in the image quality through equalization, and that GNU Radio serves as a great tool in this experimentation.

VI. Conclusion

We have presented `gr-tempest`, an open-source implementation of TEMPEST that uses GNU Radio, the most popular SDR framework. This allows, for instance, using any SDR hardware and a very easy experimentation.

In this sense, we have carefully studied the signal coming from the SDR hardware in the analog display case (i.e. VGA), derived a model and implemented some improvements to the base system that result in a much greater image quality: frequency correction and equalization. We have discussed a basic approach to the latter with very encouraging results, and several improvements are foreseeable. Maybe the most interesting alternative would be to apply a blind equalization technique [18], where the filter is automatically set and minimizes the assumptions.

Regarding digital displays, the base system may be used as-is to obtain an image as the one shown in Fig. 11. Recall that in this case 10 bits per pixel are sent, rendering equalization impractical. A promising approach, and an avenue of research we are currently exploring, is to instead learn to map from the complex-valued matrix of the spied image to the original one (in Fig. 11 we are taking its magnitude as in the vanilla case). This may be achieved through deep learning, which have obtained remarkable results precisely in image processing. There are some previous attempts in this sense that strive at, for instance, detecting letters and numbers, but take the magnitude of the complex (i.e. they use TempestSDR) [19], [20]. As we have illustrated here, this may result in an important loss of information. In fact, and in order to help with this learning, `gr-tempest` provides a GNU Radio flowgraph that simulates the spied signal.

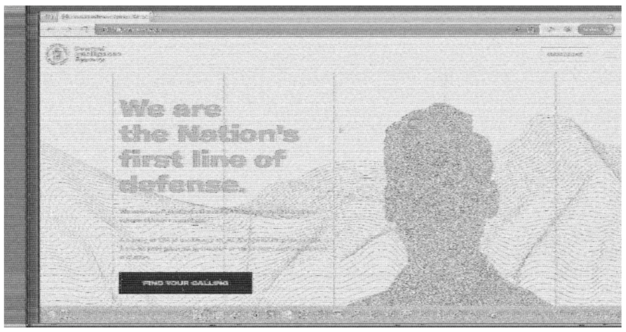


Fig. 11: An example of the spied image when using HDMI.

References

[1] J. Friedman, "Tempest: A signal problem," *NSA Cryptologic Spectrum*, vol. 35, p. 76, 1972.

[2] W. van Eck, "Electromagnetic radiation from video display units: An eavesdropping risk?" *Computers Security*, vol. 4, no. 4, pp. 269–286, 1985. [Online]. Available: https://www.sciencedirect.com/science/article/pii/016740488590046X

[3] M. G. Kuhn, "Compromising emanations: eavesdropping risks of computer displays," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-577, Dec. 2003. [Online]. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-577.pdf

[4] A. M. Wyglinski, D. P. Orofino, M. N. Ettus, and T. W. Rondeau, "Revolutionizing software defined radio: case studies in hardware, software, and education," *IEEE Communications magazine*, vol. 54, no. 1, pp. 68–75, 2016.

[5] V. González-Barbone, P. Belzarena, and F. Larroca, "Software defined radio: From theory to real world communications," in *2018 XIII Technologies Applied to Electronics Teaching Conference (TAEE)*, 2018, pp. 1–7.

[6] M. Marinov, "Remote video eavesdropping using a software-defined radio platform," *MS thesis, University of Cambridge*, 2014.

[7] Rohde & Schwartz, "R&S FSWT test receiver. TEMPEST measuring receiver with digital signal evaluation," https://www.rohde-schwarz.com/us/products/test-and-measurement/tempest-tests/rs-fswt-test-receiver_63493-310144.html.

[8] "GNU Radio. The free & open software radio ecosystem ," https://www.gnuradio.org/.

[9] "VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT)," 2013, https://glenwing.github.io/docs/VESA-DMT-1.13.pdf.

[10] "Digital Visual Interface," Digital Display Working Group, Standard, April 1999.

[11] Ettus Research, "USRP B200mini," https://www.ettus.com/all-products/usrp-b200mini/.

[12] Nuand, "bladeRF 2.0 micro xA5," https://www.nuand.com/product/bladerf-xa5/.

[13] R. G. Gallager, *Principles of digital communication*. Cambridge University Press Cambridge, UK, 2008.

[14] M. Rice, *Digital Communications: A Discrete-Time Approach (2nd edition)*, 2020.

[15] Mini-Circuits, "Low Pass Filter SLP-450+," https://www.minicircuits.com/pdfs/SLP-450+.pdf.

[16] ——, "High Pass Filter SHP-250+," https://www.minicircuits.com/pdfs/SHP-250+.pdf.

[17] ——, "Low Noise Amplifier ZX60-P103LN+," https://www.minicircuits.com/pdfs/ZX60-P103LN+.pdf.

[18] Z. Ding and Y. Li, *Blind equalization and identification*. CRC press, 2018.

[19] Z. Liu, N. Samwel, L. Weissbart, Z. Zhao, D. Lauret, L. Batina, and M. Larson, "Screen gleaning: A screen reading tempest attack on mobile devices exploiting an electromagnetic side channel," in *NDSS Symposium 2021: The Network and Distributed System Security Symposium (NDSS) 2021, 21-25 February 2021*. Sl: NDSS, 2021, pp. 1–15.

[20] F. Lemarchand, C. Marlin, F. Montreuil, E. Nogues, and M. 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)*, 2020, pp. 2882–2886.