Tempest 2.0 Revolution

Pedro C. aka s4ur0n
(@NN2ed_s4ur0n)

#CyberCamp17

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD

Tempest 2.0 Revolution

```
class PedroC:
    def __init__(self):
        self.name =       'Pedro Candel'
        self.nick =       's4ur0n'
        self.email =      's4ur0n@s4ur0n'
        self.website =    'https://www.s4ur0n.com'
        self.nickname =   '@NN2ed_s4ur0n'
        self.role =       'Security Researcher'
        self.interest =   [ 'Reversing', 'Malware',
                            'Offensive Security', '…' ]
        self.member_of = [ 'mlw.re', 'OWASP',
                           'NetXploit', '…' ]
```

Tempest

Introducción

Tempest 2.0 Revolution

El **encubrimiento de canal** (**covert channel)** es una ***técnica de evasión*** que permite a un atacante enviar información empleando un canal legítimo pero no previsto para dicho uso.

Por ejemplo, podemos enviar información en las cabeceras TCP/IP, en los campos de datos no usados, etc.

¿Qué hay acerca de **exfiltrar información** en un equipo completamente aislado donde no existe comunicación TCP/IP u otros?

Tempest 2.0 Revolution

TEMPEST es una especificación de la **NSA** y una certificación de la **OTAN** en referencia al espionaje de los sistemas de información a través de la ***fuga de emanaciones***, incluyendo las **señales de radio (emisiones electromagnéticas)**, señales eléctricas, sonidos y vibraciones.

TEMPEST cubre ambos métodos: El **espionaje** pero también **la protección** de los equipos para evitarlo.

Esta última parte se conoce como **seguridad de las emisiones (EMSEC)** dentro del conjunto de **seguridad de las comunicaciones (COMSEC)**

Tempest 2.0 Revolution

Los métodos de la **NSA** para el espionaje se encuentran ***clasificados***, pero algunos de los **estándares de protección** se publican por la propia NSA o el Departamento de Defensa.

En los estándares sobre TEMPEST, aparecen por ejemplo, la ***distancia*** de los **equipos a las paredes**, la cantidad de ***blindaje*** en **edificios y equipos**, cables de ***separación*** de distancia que transportan **materiales clasificados** o no clasificados, ***filtros*** en **cables** e incluso ***distancia y blindaje*** entre **cables/equipos**. El ***ruido*** también puede proteger la información **enmascarando los datos reales**.

Tempest 2.0 Revolution

La mayoría de **TEMPEST** se encuentra relacionado sobre la fuga de **emanaciones electromagnéticas**, pero incluye también ***sonidos*** o ***vibraciones***.

Por ejemplo, sería posible espiar las pulsaciones de un usuario empleando el ***sensor de movimiento*** del dispositivo móvil.

Estas **fugas de información** si se **interceptan y analizan**, pueden ***divulgar*** la información transmitida, recibida, manejada o procesada por cualquier equipo.

Tempest 2.0 Revolution

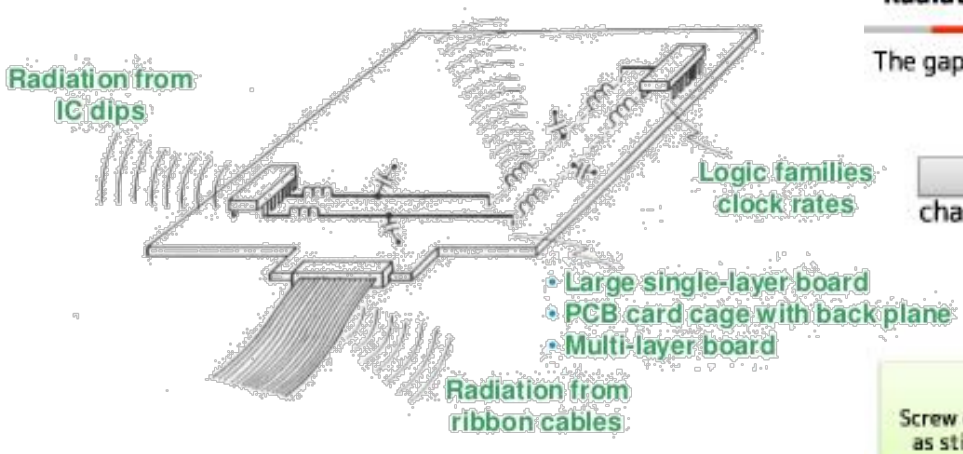
Los ataques TEMPEST se sustentan sobre el ***principio*** de que ***cualquier dispositivo electrónico*** **emite pequeñas radiaciones electromagnéticas** durante su uso normal.

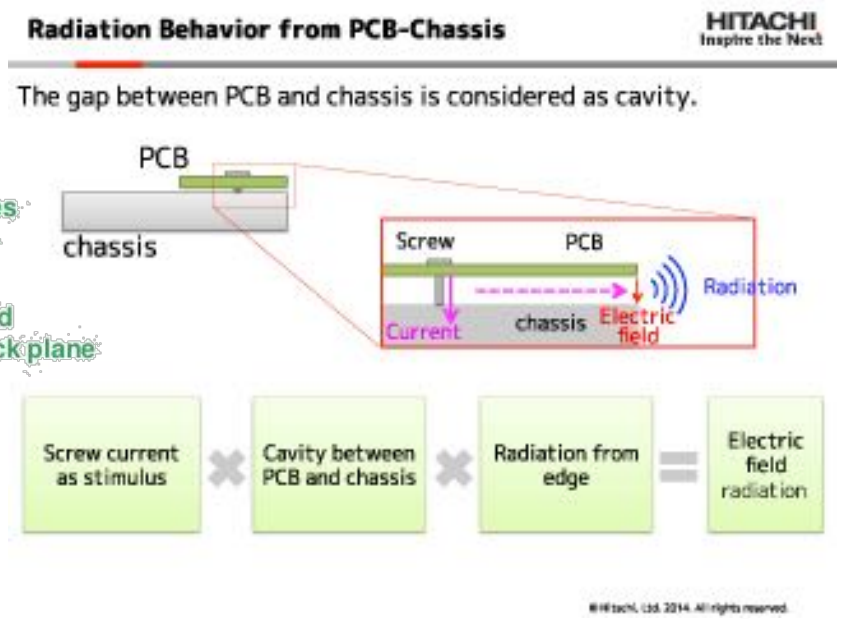
Esto sucede cada vez que una corriente eléctrica **cambia de voltaje** y genera **pulsos electromagnéticos** que se irradian como ***ondas de radio invisibles***. Estas ondas pueden alcanzar grandes distancias en situaciones ideales.

La **CPU** ahora mismo se encuentra emitiendo y sus señales **pueden ser ampliadas** por el bus del sistema a modo de **antena emisora**.

Tempest 2.0 Revolution

PRINCIPAL RADIATION SOURCES ON PRINTED CIRCUIT BOARD





Tempest 2.0 Revolution

Es un **ataque pasivo** en el sent… ***…uede ser detectado*** ya que el… conect… …a.

No p… …equipo que ir… …ro monito… …**el sistem**…

Ataques Tempest

¿Mito o realidad?

Tempest 2.0 Revolution

```
Date: Wed, 18 Aug 1999 19:15:09 +0300 (EEST)
From: Berke Durak <durakb@crit2.univ-montp2.fr>
To: cypherpunks@toad.com
Subject: Controlled CPU TEMPEST emanations

Hello,

After having implemented and successfully tested Ross Anderson's idea
to use the video output to synthesize a mediumwave AM signal, I
wondered if a similar effect could be obtained by using only the CPU,
since it was easy to correlate CPU activity with radio noise. I've
just written a quick C program that tries to force activity on the
memory bus in a repetitive pattern, with adjustable frequency. After
having fiddled with the timings for about one hour, I managed to
broadcast a test tune using my Pentium 120 running Linux, giving
extremely clear reception on FM band at about 87.5 Mhz (I have in no
way calculated or predicted this frequency).

Be warned that my understanding of radio waves is bad and incomplete,
and that I have no particular radio equipment, save a walkman and a
radio cassette player.

I found that it is possible to hear the test tune over the whole
"consumer" medium- and short-wave spectrum (530-1600 KHz, 2.3-22 MHz)
using the walkman, which has a digital synthesized PLL radio
(which is generally very sensitive to electrical noise), provided the
radio is held at a distance of less than two meters around the CPU,
which suggests that there are spectral components of CPU activity at
many frequencies dividing the clock frequency and at their harmonics
(which gives a very rich spectrum).  The reception in the FM band is
much more clean, and it is possible to hear the test tune in the next
room (three to four meters).

I've found that accesses to the main memory create much more noise
that other CPU activity, which is readily understandable. As it is
```

Source: https://cryptome.org/tempest-cpu.htm

Tempest 2.0 Revolution

```
#include <math.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>
#include <stdio.h>
#include <SDL.h>
#include <string.h>
```

```
Tempest for Eliza - by erikyyy !
--------------------------------

Read the README file to understand what's happening
if you do not read it, you will NOT know what to do

Pixel Clock 105000000 Hz
X Resolution 1024 Pixels
Y Resolution 768 Pixels
Horizontal Total 1400 Pixels
AM Carrier Frequency 10000000 Hz
```

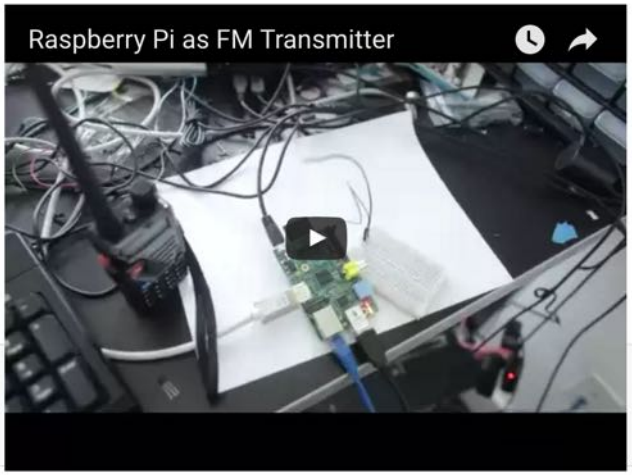
```
  };
  while (tv_usec>=1000000) {
tv_usec-=1000000;
tv_sec++;
  };
};
bool zero()
```

Source: http://www.erikyyy.de/tempest/

Tempest 2.0 Revolution

Turning the Raspberry Pi Into an FM Transmitter

Contents [hide]



Steps to play sound:

(Created by Oliver Mattos and Oskar Weigl. Code is GPL)

```
sudo python
>>> import PiFm
>>> PiFm.play_sound("sound.wav")
```

Now connect a 70cm (optimally, ~20cm will do) or so plain wire to GPIO 4 (which is pin 7 on header P1) to act as an antenna, and tune an FM radio to 103.3Mhz.

Download the module here:

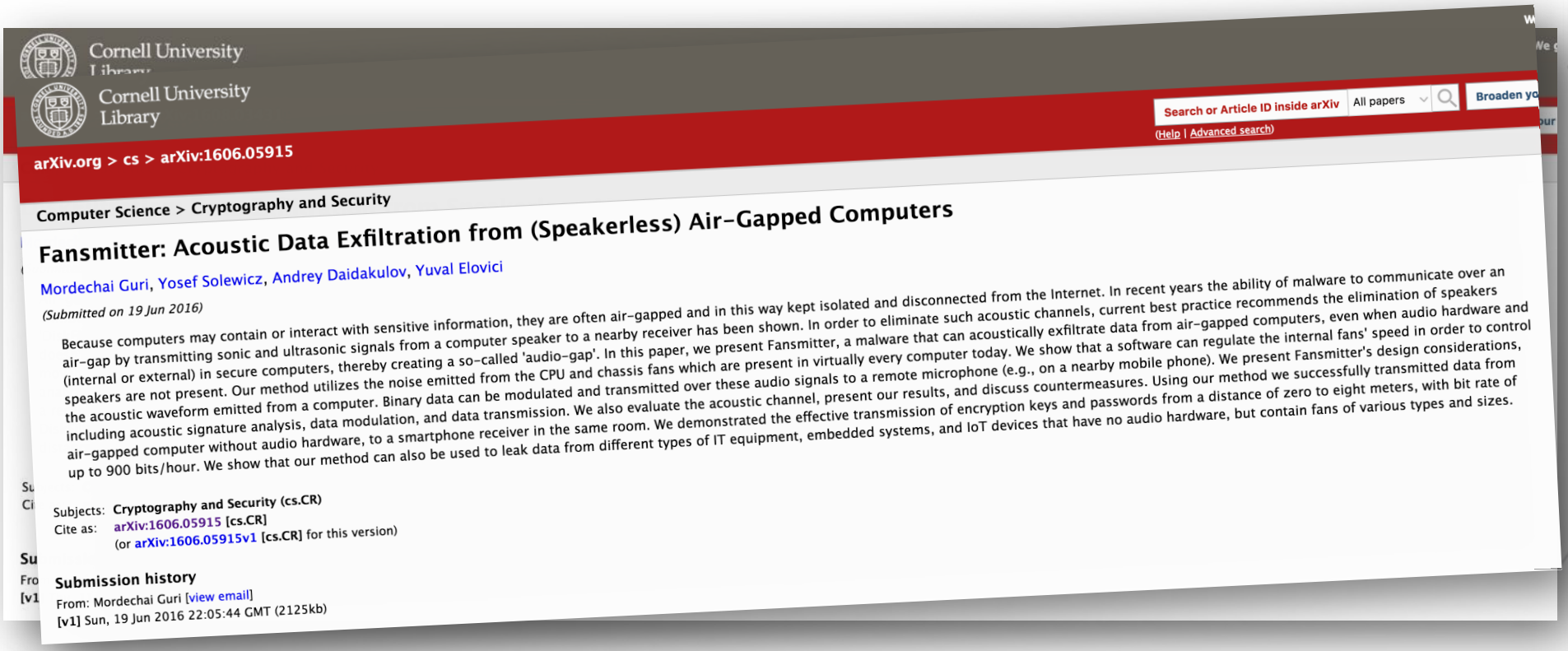
- [Download Now!]

(this contains both source and a ready to go binary. Just run the above code in the same folder. The antenna is optional, but range is reduced from ~100 meters to ~10cm without the antenna. Th format.)

New! Now with stereo

Source: http://www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter

Tempest 2.0 Revolution



Cornell University Library

arXiv.org > cs > arXiv:1606.05915

Search or Article ID inside arXiv | All papers | (Help | Advanced search)

Computer Science > Cryptography and Security

Fansmitter: Acoustic Data Exfiltration from (Speakerless) Air-Gapped Computers

Mordechai Guri, Yosef Solewicz, Andrey Daidakulov, Yuval Elovici

(Submitted on 19 Jun 2016)

Because computers may contain or interact with sensitive information, they are often air-gapped and in this way kept isolated and disconnected from the Internet. In recent years the ability of malware to communicate over an air-gap by transmitting sonic and ultrasonic signals from a computer speaker to a nearby receiver has been shown. In order to eliminate such acoustic channels, current best practice recommends the elimination of speakers (internal or external) in secure computers, thereby creating a so-called 'audio-gap'. In this paper, we present Fansmitter, a malware that can acoustically exfiltrate data from air-gapped computers, even when audio hardware and speakers are not present. Our method utilizes the noise emitted from the CPU and chassis fans which are present in virtually every computer today. We show that a software can regulate the internal fans' speed in order to control the acoustic waveform emitted from a computer. Binary data can be modulated and transmitted over these audio signals to a remote microphone (e.g., on a nearby mobile phone). We present Fansmitter's design considerations, including acoustic signature analysis, data modulation, and data transmission. We also evaluate the acoustic channel, present our results, and discuss countermeasures. Using our method we successfully transmitted data from air-gapped computer without audio hardware, to a smartphone receiver in the same room. We demonstrated the effective transmission of encryption keys and passwords from a distance of zero to eight meters, with bit rate of up to 900 bits/hour. We show that our method can also be used to leak data from different types of IT equipment, embedded systems, and IoT devices that have no audio hardware, but contain fans of various types and sizes.

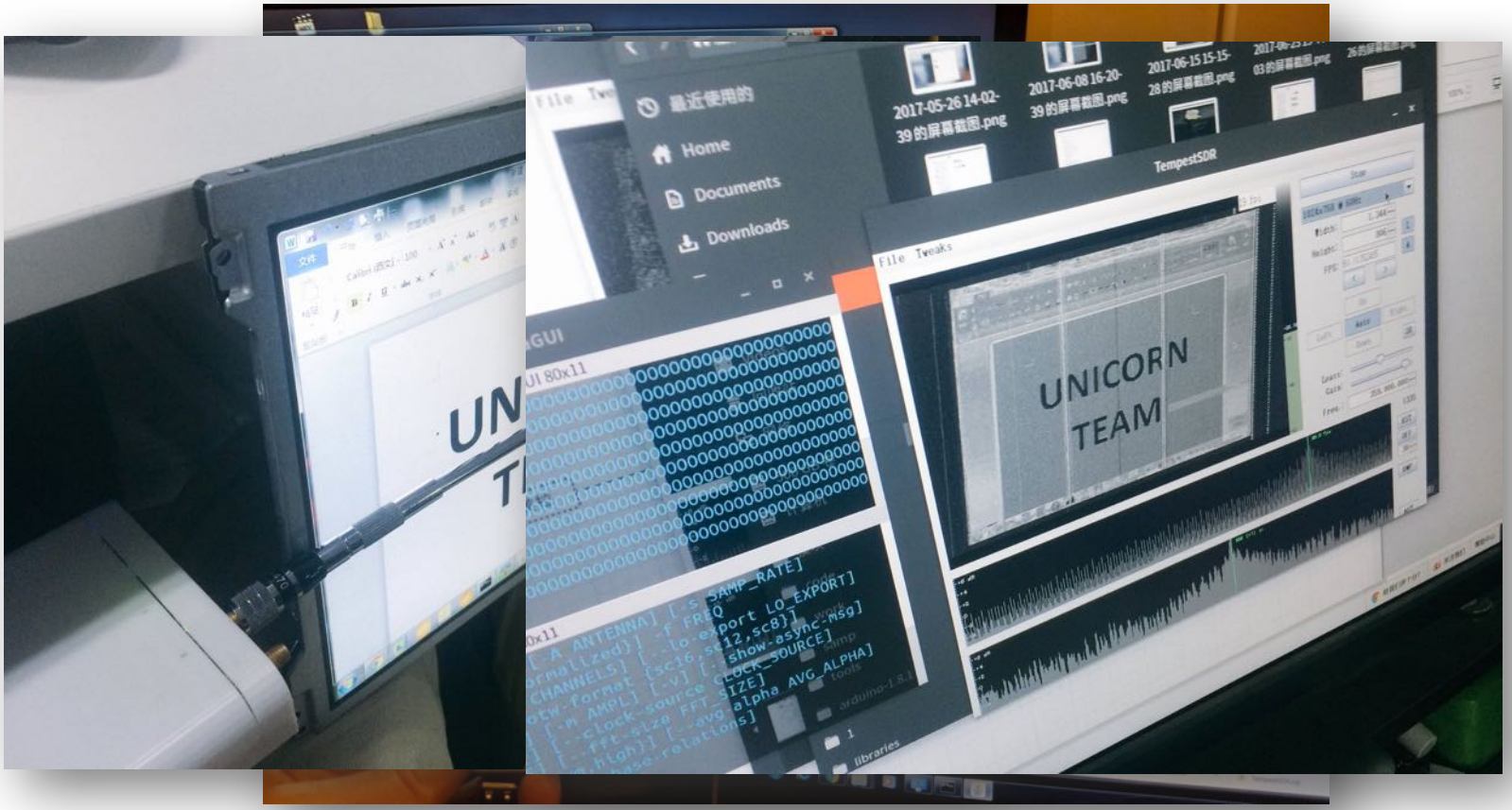
Subjects: Cryptography and Security (cs.CR)
Cite as: arXiv:1606.05915 [cs.CR]
(or arXiv:1606.05915v1 [cs.CR] for this version)

Submission history
From: Mordechai Guri [view email]
[v1] Sun, 19 Jun 2016 22:05:44 GMT (2125kb)

Source(s): https://arxiv.org/abs/1608.03431 &
https://arxiv.org/abs/1606.05915

Tempest 2.0 Revolution

Señales del monitor con un SDR:



Source(s): https://github.com/martinmarinov/TempestSDR & https://www.rtl-sdr.com/tempestsdr-a-sdr-tool-for-eavesdropping-on-computer-screens-via-unintentionally-radiated-rf/


Tempest Revolution 2.0
Nuevo canal encubierto
CañApp

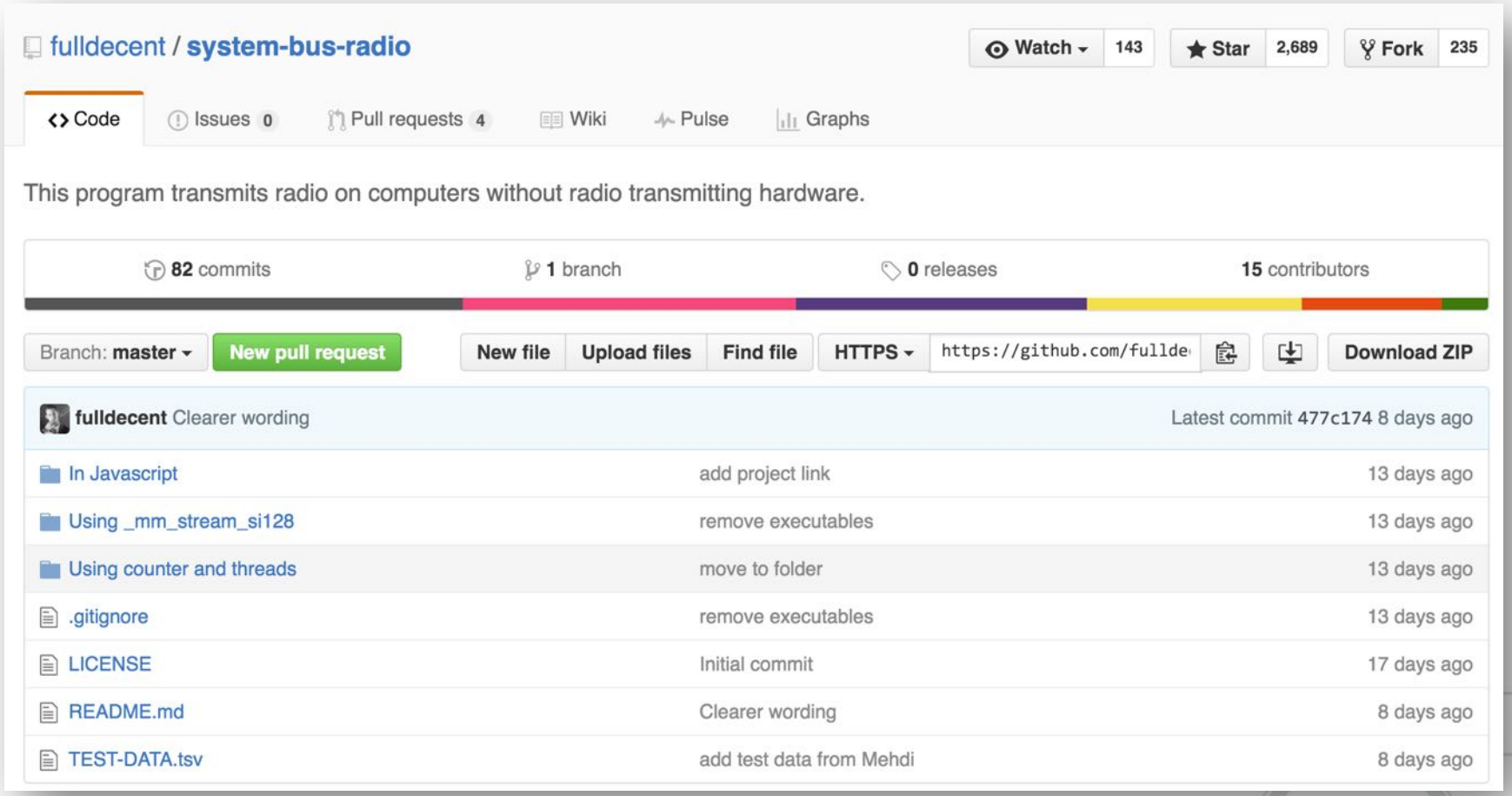
Tempest 2.0 Revolution

Objetivo:

Emplear un **cover channel legítimo** y que no pueda ser anulado en sistemas aislados para **provocar fugas intencionadas de información**.

Tempest 2.0 Revolution

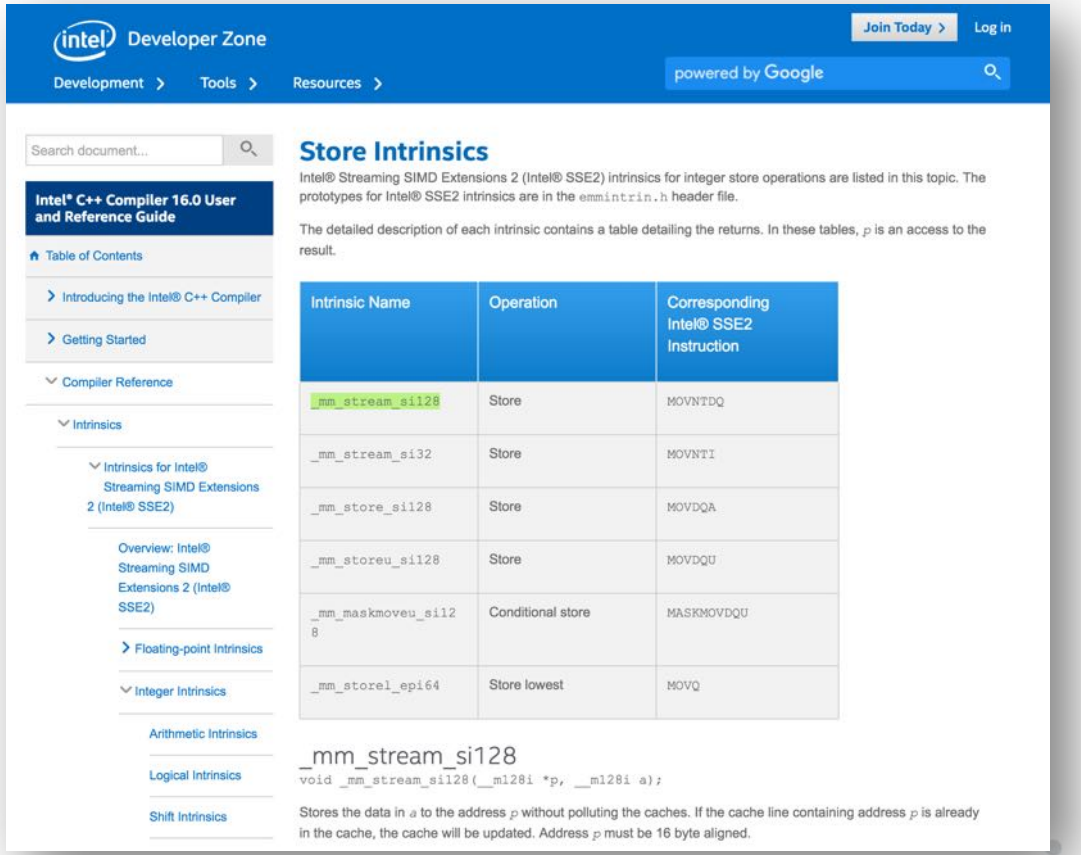
System Bus Radio: proyecto inicial



Source: https://github.com/fulldecent/system-bus-radio

Tempest 2.0 Revolution

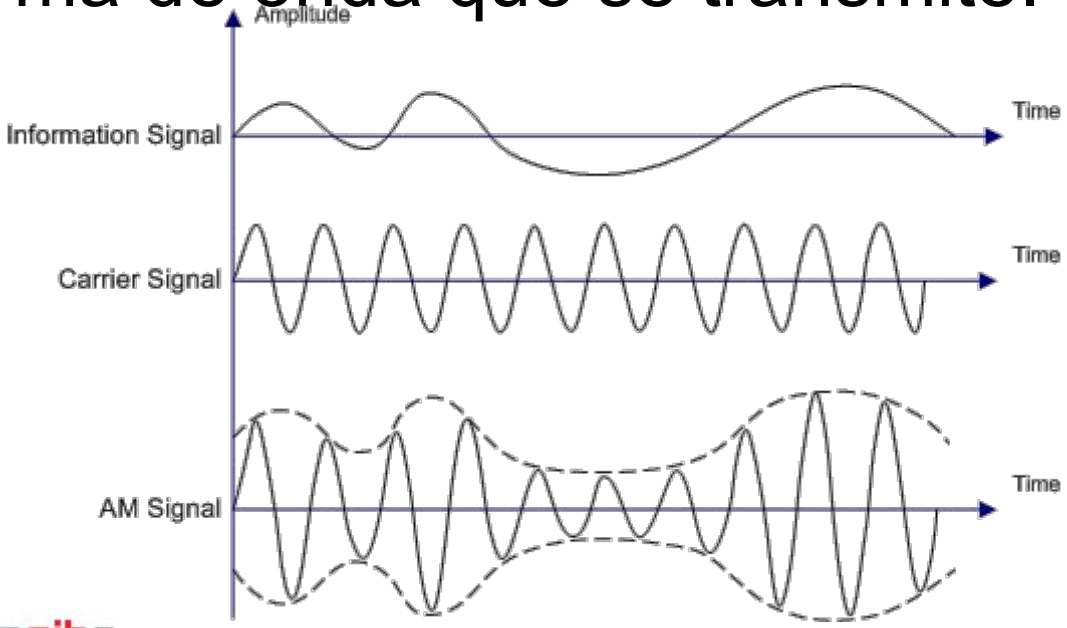
¿Cómo? Ejecutar **instrucciones legítimas** en el equipo que **causen radiaciones electromagnéticas** por el ***ruido generado***.

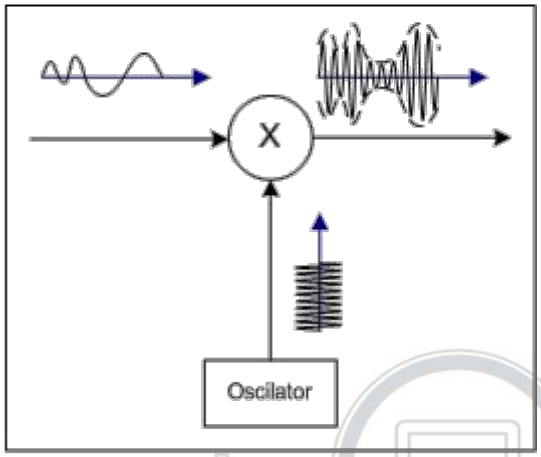


Tempest 2.0 Revolution

Transmitir información a través de una ***onda portadora de radio***.

En **amplitud modulada**, la amplitud (***intensidad de señal***) de la onda portadora varía en proporción a la forma de onda que se transmite.





Tempest 2.0 Revolution

The actual emissions are caused by the `_mm_stream_si128` instruction that writes through to a memory address. Inspiration for using this instruction was provided in:

> Guri, M., Kachlon, A., Hasson, O., Kedma, G., Mirsky, Y. and Elovici, Y., 2015. GSMem: data exfiltration from air-gapped computers over GSM frequencies. In 24th USENIX Security Symposium (USENIX Security 15) (pp. 849-864).

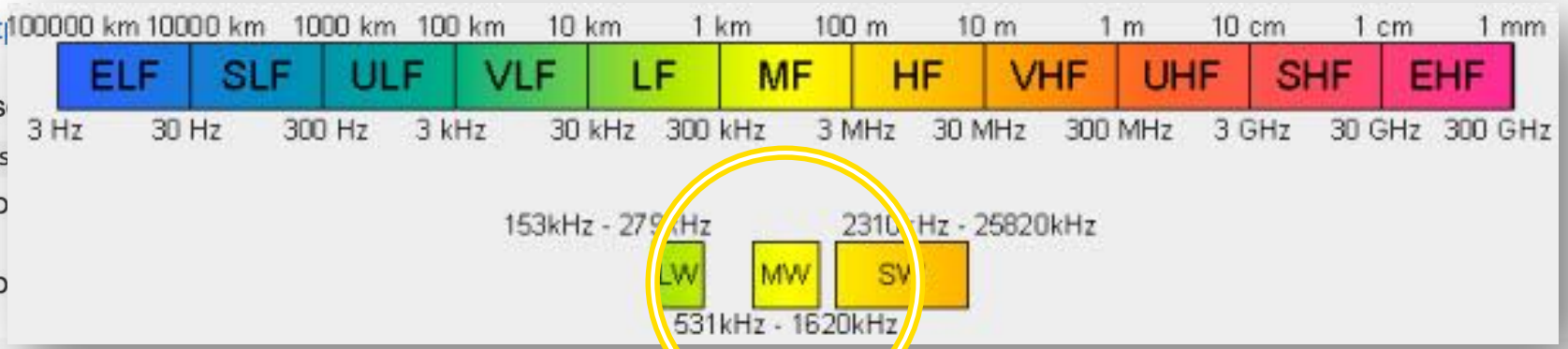
htt

Pleas

`_mm_s`

impro

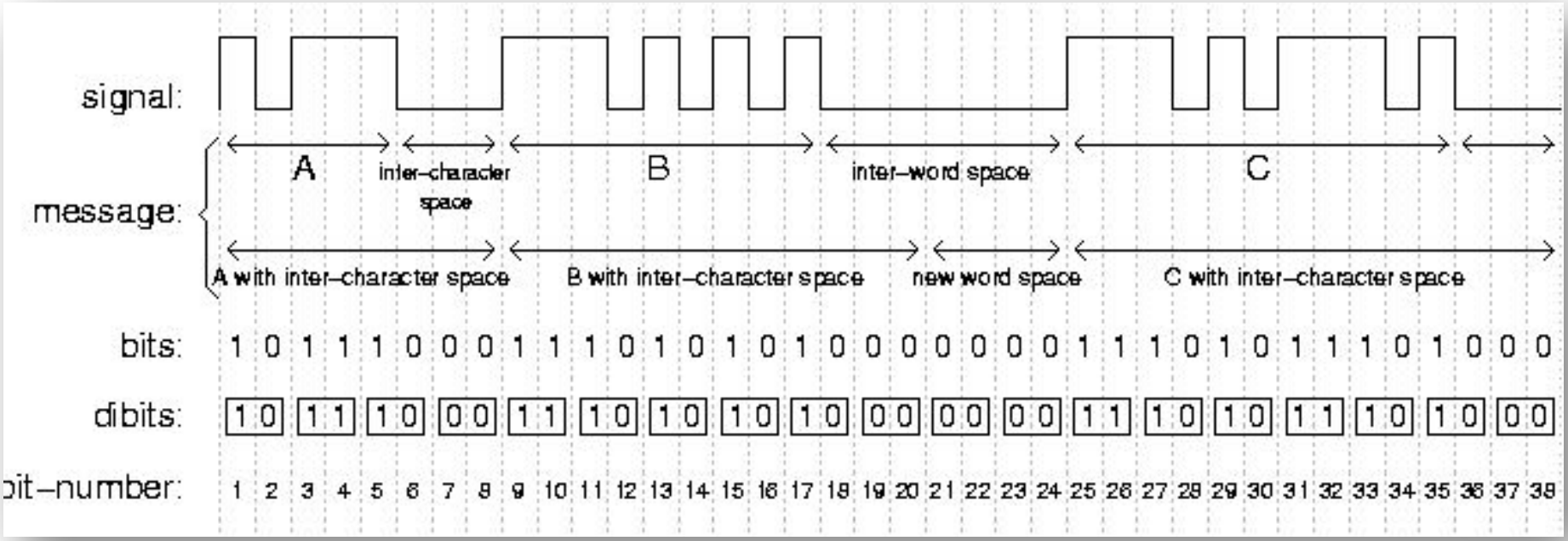
The p



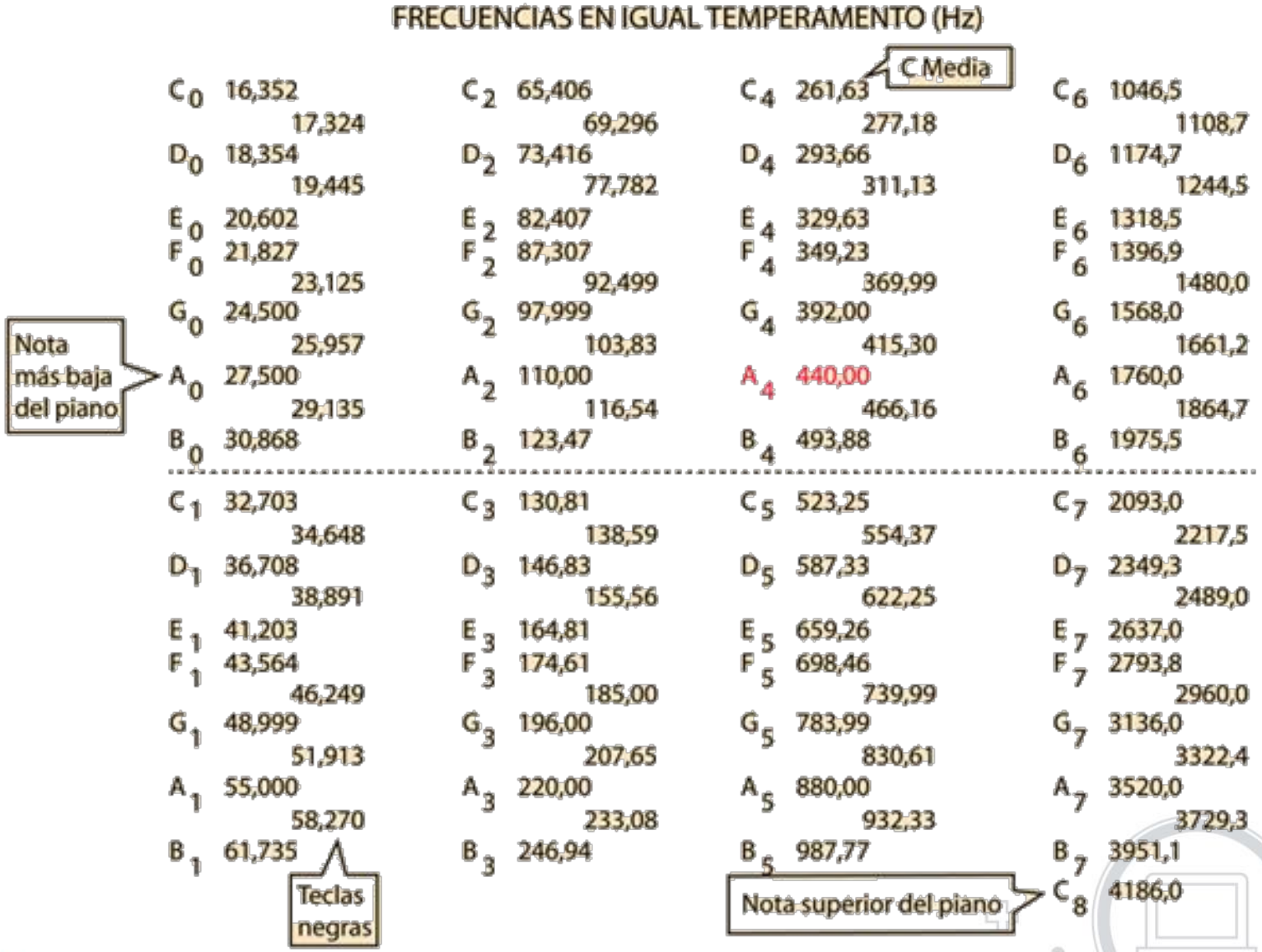

```
|<--------------------TIME--------------------->|
|                                               |
|¯|_|¯|_|¯|_____________|¯|_|¯|_|¯|_____________
|                       |   |   |
|<------SIGNAL--------->|   |   |
                        |   |
                        |<->| CARRIER
```

Tempest 2.0 Revolution

Anteriormente se realizaba en **CODIGO MORSE** empleando una serie de transformaciones.



Tempest 2.0 Revolution



C_0	16,352	C_2	65,406	C_4	261,63	C_6	1046,5
	17,324		69,296		277,18		1108,7
D_0	18,354	D_2	73,416	D_4	293,66	D_6	1174,7
	19,445		77,782		311,13		1244,5
E_0	20,602	E_2	82,407	E_4	329,63	E_6	1318,5
F_0	21,827	F_2	87,307	F_4	349,23	F_6	1396,9
	23,125		92,499		369,99		1480,0
G_0	24,500	G_2	97,999	G_4	392,00	G_6	1568,0
	25,957		103,83		415,30		1661,2
A_0	27,500	A_2	110,00	A_4	440,00	A_6	1760,0
	29,135		116,54		466,16		1864,7
B_0	30,868	B_2	123,47	B_4	493,88	B_6	1975,5
C_1	32,703	C_3	130,81	C_5	523,25	C_7	2093,0
	34,648		138,59		554,37		2217,5
D_1	36,708	D_3	146,83	D_5	587,33	D_7	2349,3
	38,891		155,56		622,25		2489,0
E_1	41,203	E_3	164,81	E_5	659,26	E_7	2637,0
F_1	43,564	F_3	174,61	F_5	698,46	F_7	2793,8
	46,249		185,00		739,99		2960,0
G_1	48,999	G_3	196,00	G_5	783,99	G_7	3136,0
	51,913		207,65		830,61		3322,4
A_1	55,000	A_3	220,00	A_5	880,00	A_7	3520,0
	58,270		233,08		932,33		3729,3
B_1	61,735	B_3	246,94	B_5	987,77	B_7	3951,1
						C_8	4186,0

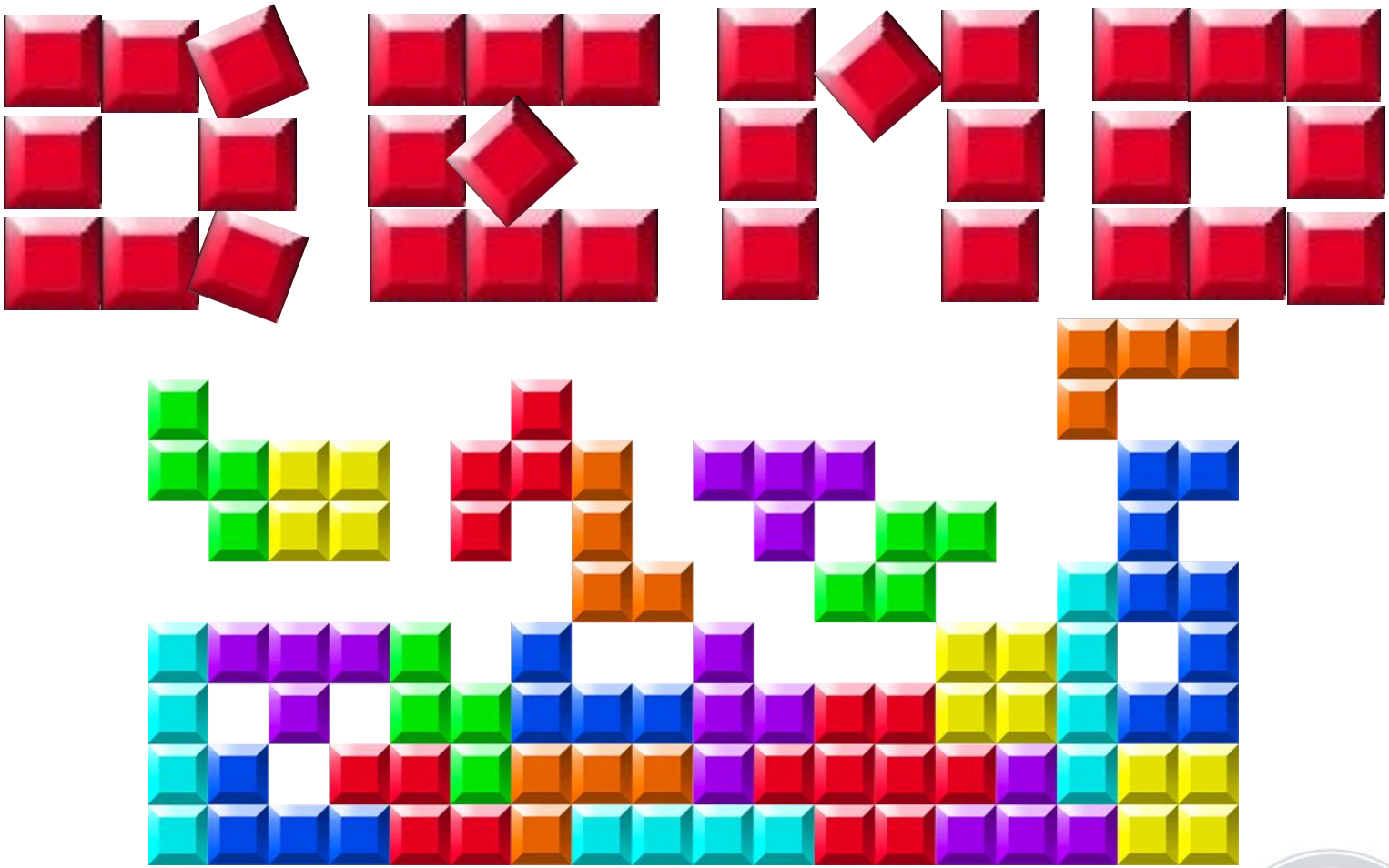
Tempest 2.0 Revolution

```
void Beep(float frequency, float time) {
    time = (time / 1000);
    square_am_signal(time, frequency);
}

void Sleep(float time) {
    time = (time / 1000);
    square_am_signal(time, 0);
}
```

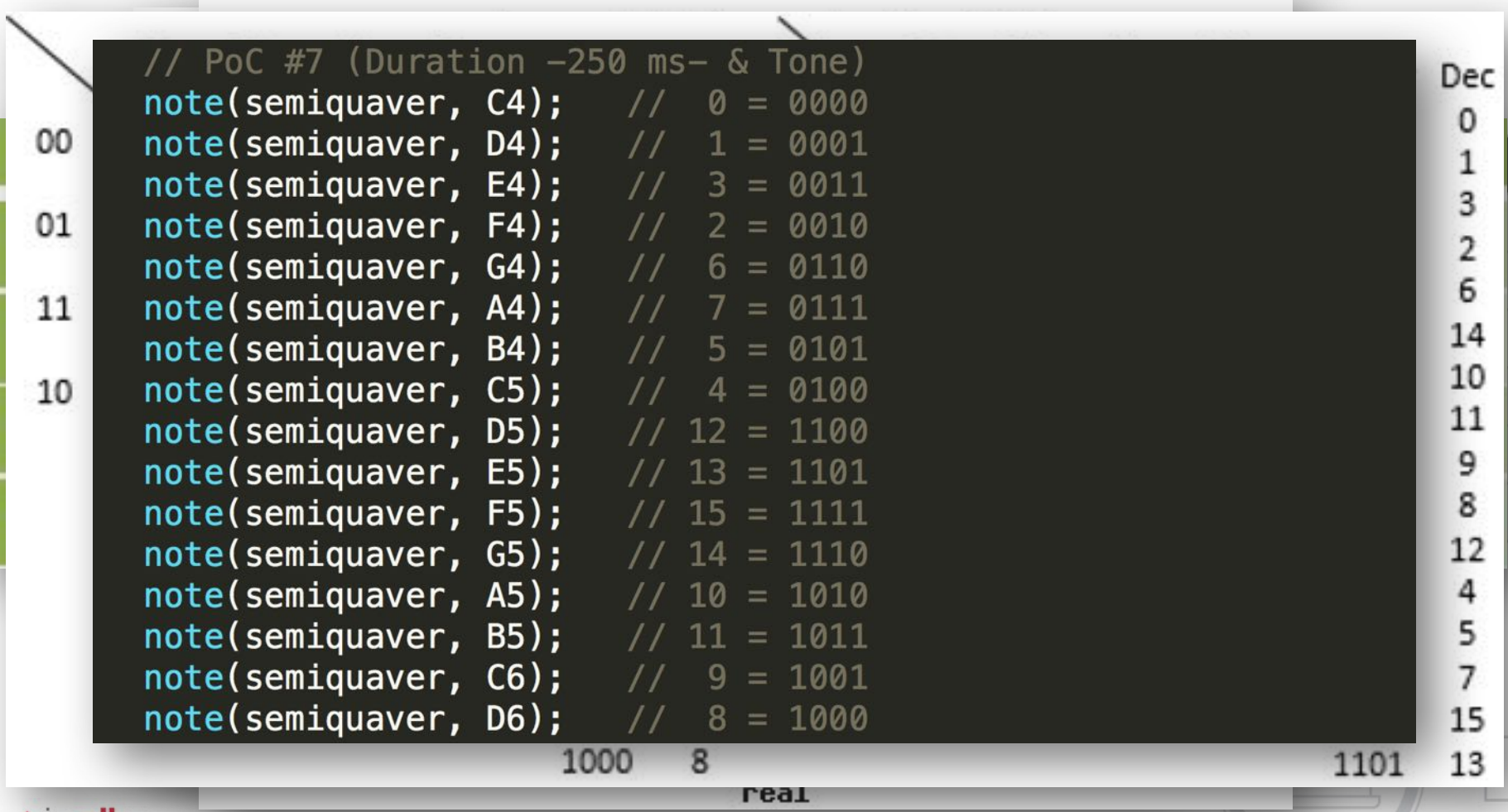
```
reg_one = _mm_set_epi32(-1, -1, -1, -1);
```

Tempest 2.0 Revolution



Tempest 2.0 Revolution

Símbolos. 16QAM vs 16PSK. BER. Codificación, constelación. Código gris.



Tempest 2.0 Revolution

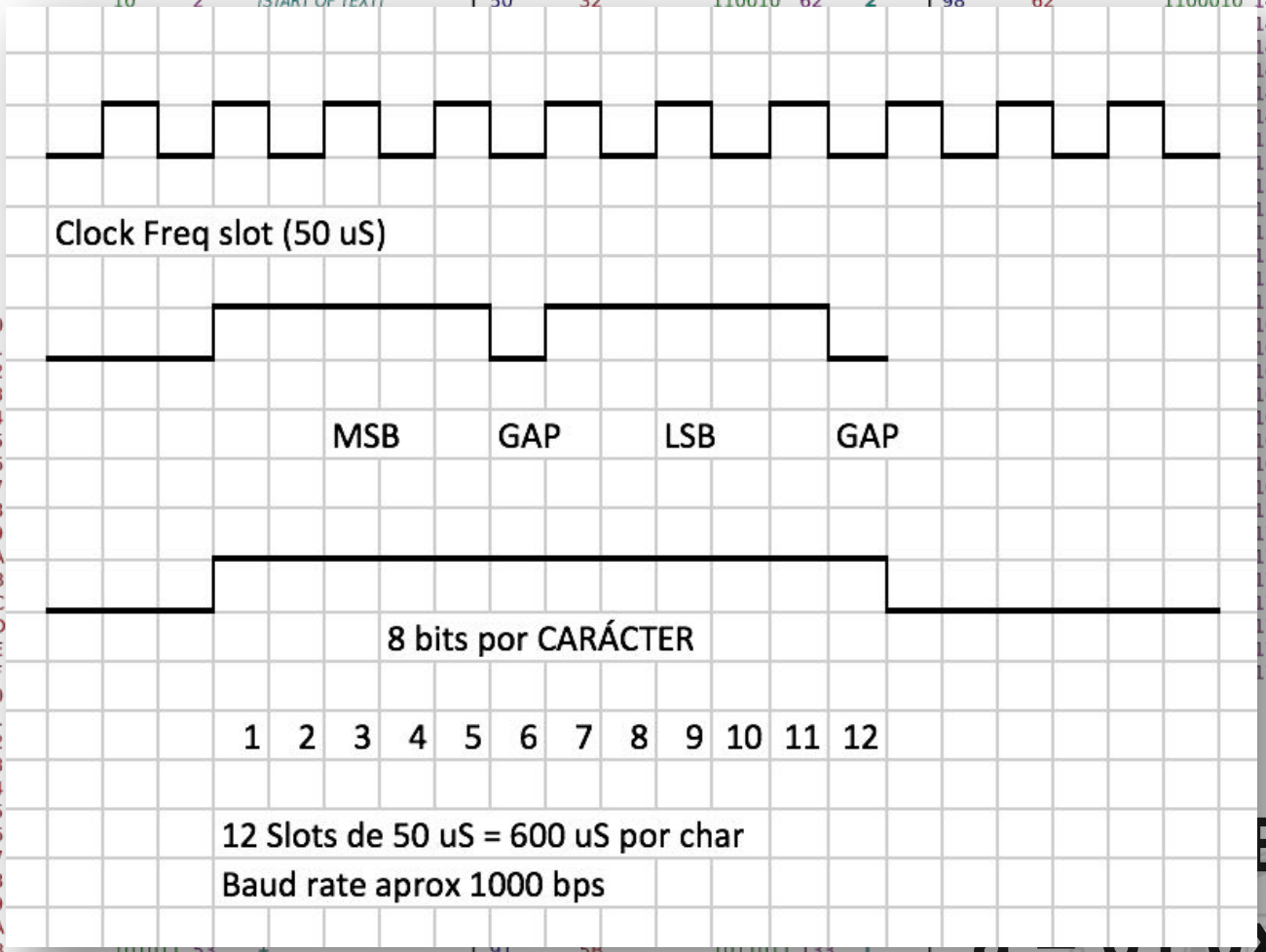
Tempest 2.0 Revolution

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3			
4	4			
5	5			
6	6			
7	7			
8	8			
9	9			
10	A			
11	B			
12	C			
13	D			
14	E			
15	F			
16	10			
17	11			
18	12			
19	13			
20	14			
21	15			
22	16			
23	17			
24	18			
25	19			
26	1A			
27	1B			
28	1C			
29	1D			
30	1E			
31	1F			
32	20			
33	21			
34	22			
35	23			
36	24			
37	25			
38	26			
39	27			
40	28			
41	29			
42	2A			
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/

Decimal	Hexadecimal	Binary	Octal	Char
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_

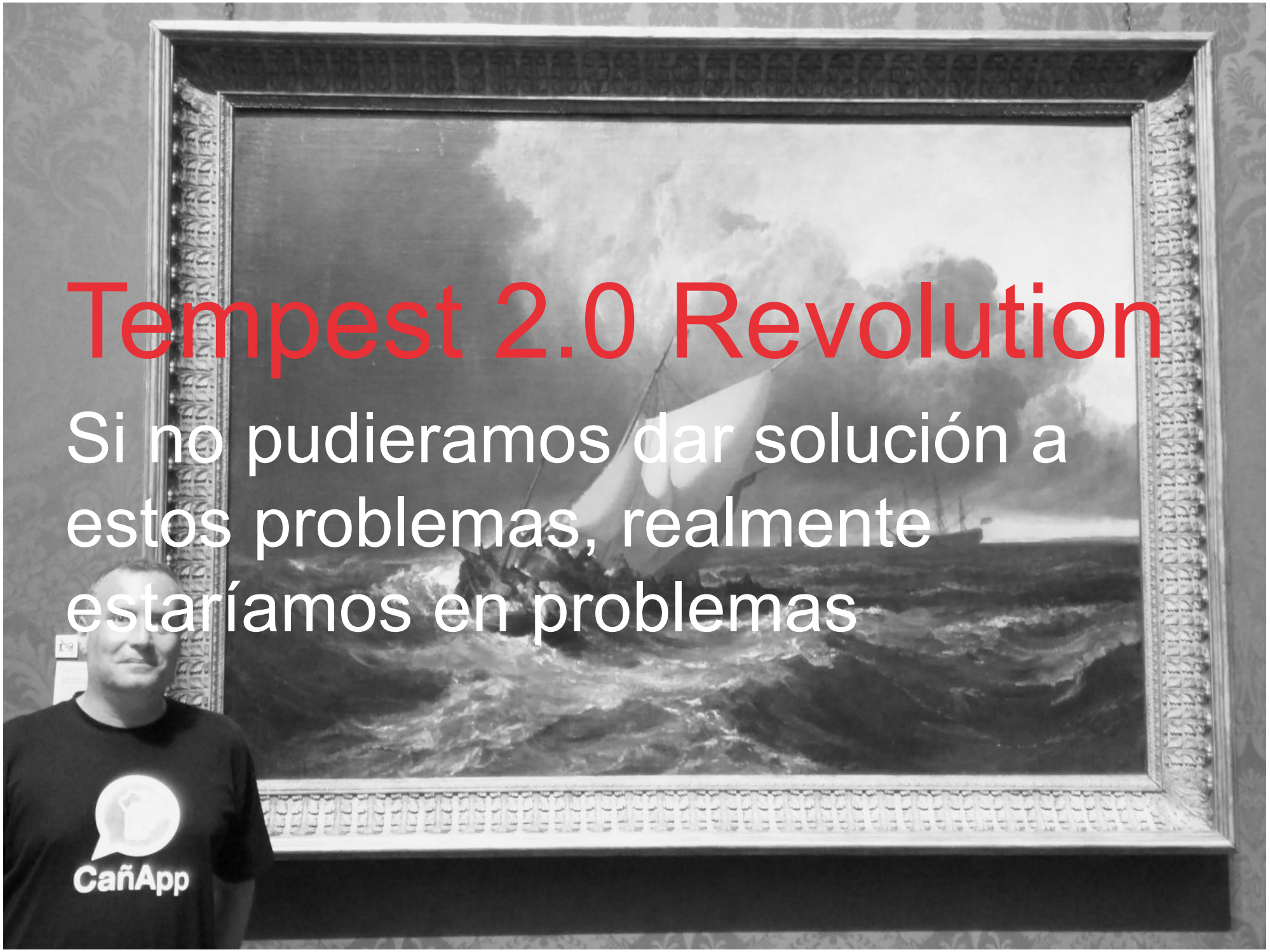
Decimal	Hexadecimal	Binary	Octal	Char
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
			143	c
			144	d
			145	e
			146	f
			147	g
			150	h
			151	i
			152	j
			153	k
			154	l
			155	m
			156	n
			157	o
			160	p
			161	q
			162	r
			163	s
			164	t
			165	u
			166	v
			167	w
			170	x
			171	y
			172	z
			173	{
			174	|
			175	}
			176	~
			177	[DEL]



B / LSB

x6 0x1

Tempest 2.0 Revolution



Tempest 2.0 Revolution

Si no pudieramos dar solución a estos problemas, realmente estaríamos en problemas

Tempest 2.0 Revolution

Houston, **we have a problem**

Transmisiones en Broadcast ☹

Criptoanálisis de las frecuencias ☹

Ficheros binarios ☹

Conjunto limitado de caracteres ☹

Tempest 2.0 Revolution

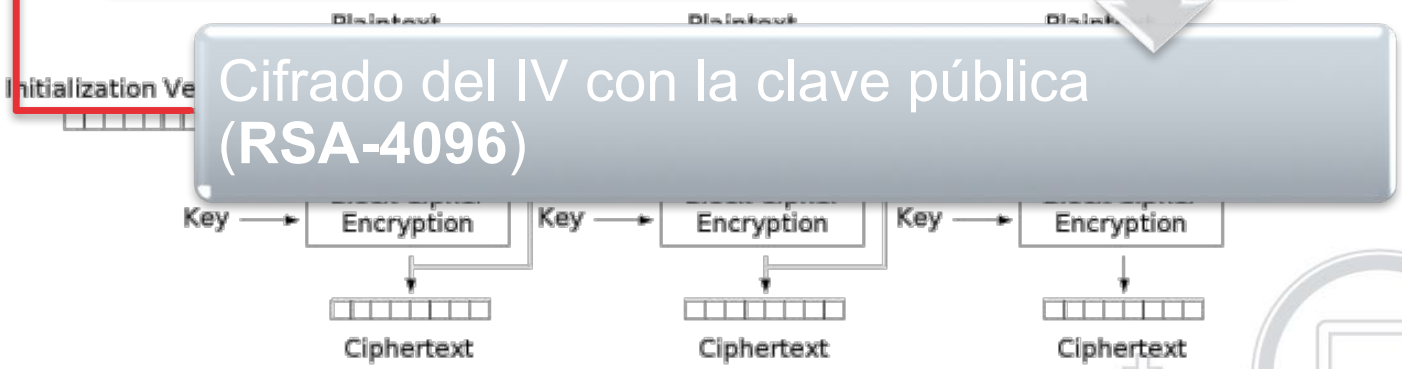
Cifrado

Extracción de la **clave pública**

Generación de un **IV**

Cifrado con el IV generado del fichero en modo **AES256-CBC**

Cifrado del IV con la clave pública (**RSA-4096**)



Cipher Block Chaining (CBC) mode encryption

Codificación

Codificación

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Tempest 2.0 Revolution

Normalización

Normalización (**su**

```
Paragraph: '+-:
-[']-------------------------------
[+] Int: 39 | MSB: 0x2 | LSB: 0x7
Playing / 0.500 seconds /  349 Hz
Playing / 0.100 seconds /    0 Hz
Playing / 0.500 seconds /  440 Hz
Playing / 0.100 seconds /    0 Hz
-[+]-------------------------------
[+] Int: 43 | MSB: 0x2 | LSB: 0x11
Playing / 0.500 seconds /  349 Hz
Playing / 0.100 seconds /    0 Hz
Playing / 0.500 seconds /  988 Hz
Playing / 0.100 seconds /    0 Hz
-[-]-------------------------------
[+] Int: 45 | MSB: 0x2 | LSB: 0x13
Playing / 0.500 seconds /  349 Hz
Playing / 0.100 seconds /    0 Hz
Playing / 0.500 seconds /  659 Hz
Playing / 0.100 seconds /    0 Hz
-[:]-------------------------------
[+] Int: 58 | MSB: 0x3 | LSB: 0x10
Playing / 0.500 seconds /  330 Hz
Playing / 0.100 seconds /    0 Hz
Playing / 0.500 seconds /  880 Hz
Playing / 0.100 seconds /    0 Hz
```

Cambio		Morse Code
Minúsculas	Inse Car	`- - . . .` + toupper(char)
Signo (+)	Mer	`. - . - . -`
Espacio entre ficheros	Apó	`. - - - - .`

Tempest 2.0 Revolution

Re

-
-

htn

k.

Discrete Fourier transforms (scipy.fftpack)

Fast Fourier Transforms (FFTs)

fft(x[, n, axis, overwrite_x])	Return discrete Fourier transform of real or complex sequence.
ifft(x[, n, axis, overwrite_x])	Return discrete inverse Fourier transform of real or complex sequence.
fft2(x[, shape, axes, overwrite_x])	2-D discrete Fourier transform.
ifft2(x[, shape, axes, overwrite_x])	2-D discrete inverse Fourier transform of real or complex sequence.
fftn(x[, shape, axes, overwrite_x])	Return multidimensional discrete Fourier transform.
ifftn(x[, shape, axes, overwrite_x])	Return inverse multi-dimensional discrete Fourier transform of arbitrary type sequence x.
rfft(x[, n, axis, overwrite_x])	Discrete Fourier transform of a real sequence.
irfft(x[, n, axis, overwrite_x])	Return inverse discrete Fourier transform of real sequence x.
dct(x[, type, n, axis, norm, overwrite_x])	Return the Discrete Cosine Transform of arbitrary type sequence x.
idct(x[, type, n, axis, norm, overwrite_x])	Return the Inverse Discrete Cosine Transform of an arbitrary type sequence.
dctn(x[, type, shape, axes, norm, overwrite_x])	Return multidimensional Discrete Cosine Transform along the specified axes.
idctn(x[, type, shape, axes, norm, overwrite_x])	Return multidimensional Discrete Cosine Transform along the specified axes.
dst(x[, type, n, axis, norm, overwrite_x])	Return the Discrete Sine Transform of arbitrary type sequence x.
idst(x[, type, n, axis, norm, overwrite_x])	Return the Inverse Discrete Sine Transform of an arbitrary type sequence.
dstn(x[, type, shape, axes, norm, overwrite_x])	Return multidimensional Discrete Sine Transform along the specified axes.
idstn(x[, type, shape, axes, norm, overwrite_x])	Return multidimensional Discrete Sine Transform along the specified axes.

Differential and pseudo-differential operators

diff(x[, order, period, _cache])	Return k-th derivative (or integral) of a periodic sequence x.
tilbert(x, h[, period, _cache])	Return h-Tilbert transform of a periodic sequence x.
itilbert(x, h[, period, _cache])	Return inverse h-Tilbert transform of a periodic sequence x.
hilbert(x[, _cache])	Return Hilbert transform of a periodic sequence x.
ihilbert(x)	Return inverse Hilbert transform of a periodic sequence x.
cs_diff(x, a, b[, period, _cache])	Return (a,b)-cosh/sinh pseudo-derivative of a periodic sequence.
sc_diff(x, a, b[, period, _cache])	Return (a,b)-sinh/cosh pseudo-derivative of a periodic sequence x.
ss_diff(x, a, b[, period, _cache])	Return (a,b)-sinh/sinh pseudo-derivative of a periodic sequence x.
cc_diff(x, a, b[, period, _cache])	Return (a,b)-cosh/cosh pseudo-derivative of a periodic sequence.
shift(x, a[, period, _cache])	Shift periodic sequence x by a: y(u) = x(u+a).

Tempest 2.0 Revolution

OSX High Sierra EPIC fail

/usr/
/Lib
/tmp
adm

Not are
encr ***t be***
cop

Tempest 2.0 Revolution

DEMO

Gracias por su atención

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD