
security is not an island
HILTONMALTA
24th Annual
FIRST
Conference
MALTA
17 - 22 June 2012
FIRST
Improving Security Together

Cryptanalysis of malware encrypted output files

Nelson Uto
CPqD

24th Annual FIRST Conference
MALTA
17 - 22 June 2012

Agenda

- Introduction.
- Cryptanalysis of File #1.
- Cryptanalysis of File #2.
- Cryptanalysis of File #3.

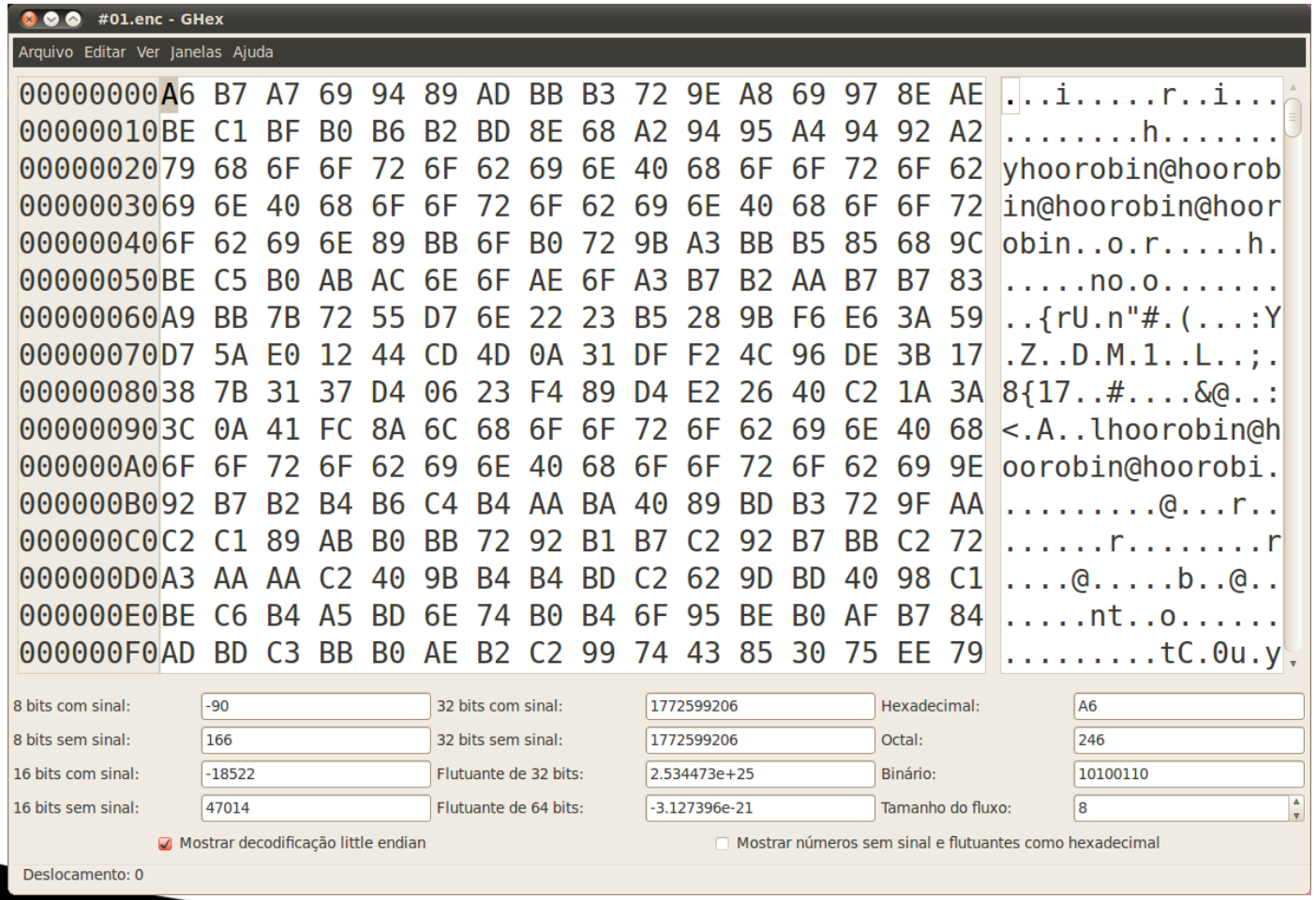
Introduction

- CPqD was hired by a big Brazilian company to find out which information had been stolen by three different malwares, that infected its environment.
- Each one of them stored information in encrypted form using different mechanisms.
- We did only have access to the encrypted files and the malware binaries, meaning we could not use the special purpose hardware targeted by them.
- Due to the sensitivity of the stolen data and signed NDA, this talk will not use the real information we retrieved from those files.

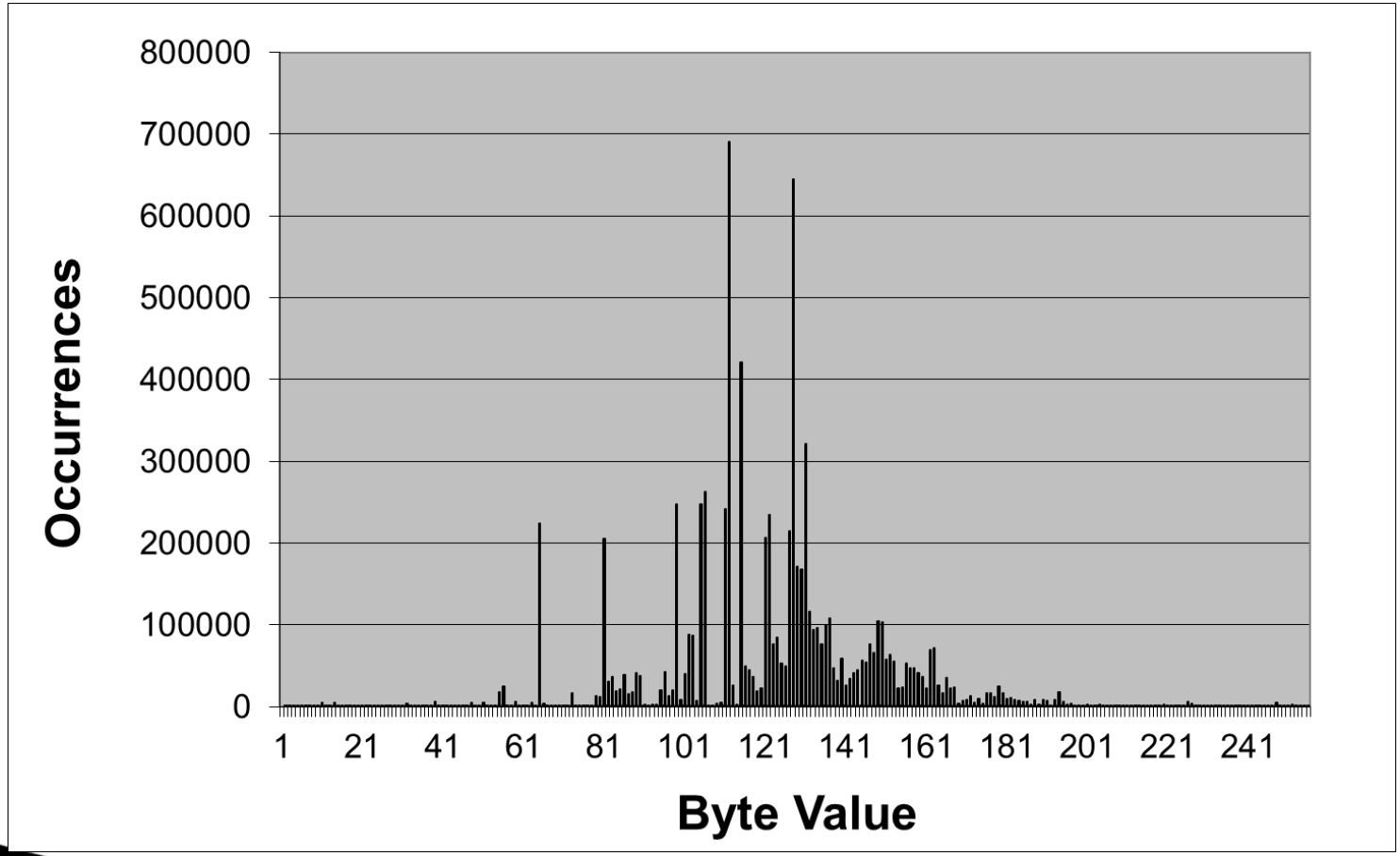
Covered topics

- Detection of weak cryptosystems.
- Cryptanalysis of classical algorithms.
- Block ciphers.
- DES.
- Modes of operation.
- Searching key in malware binary or in memory.
- Worst scenario.

File #1 – Sample



File #1 – Histogram



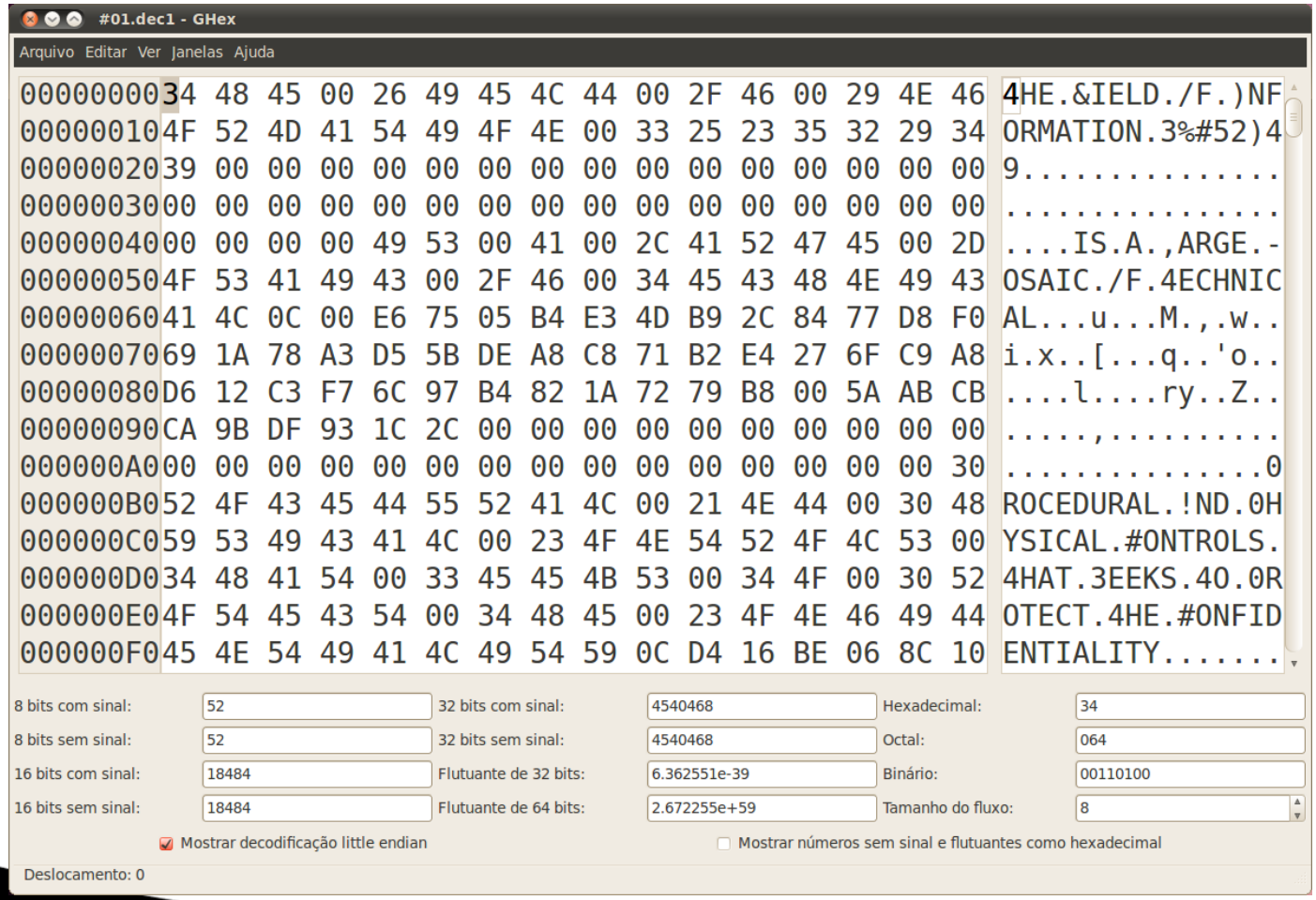
File #1 – Important facts

- File#1 is pretty redundant.
  - This means a weak cryptosystem was used.
- The distance between occurrences of the string "robin@hoo" is always multiple of its length.
- Most of the bytes has values between 80 and 180.

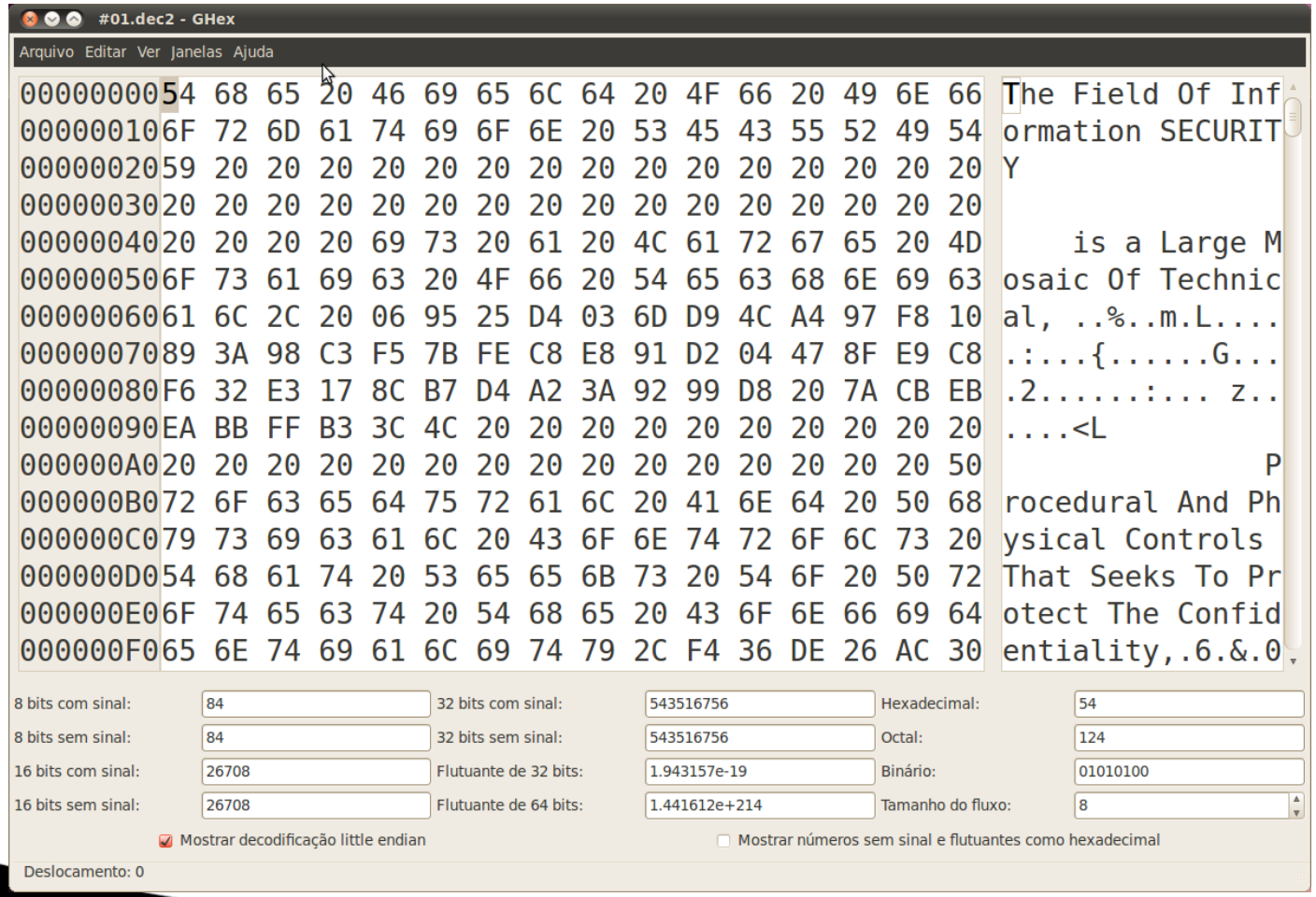
File #1 – Hypothesis

- **Hypothesis #1:** a constant number is added to each byte modulo 256 and a given string is repeated several times in the plain text.
  - Not likely, but it should be tested.
  - How?
- **Hypothesis #2:** a Vigenère cipher over an alphabet of 256 elements and period equals 9 was used.
  - Candidate key: robin@hoo

File #1 – First attempt



File #1 – Correction

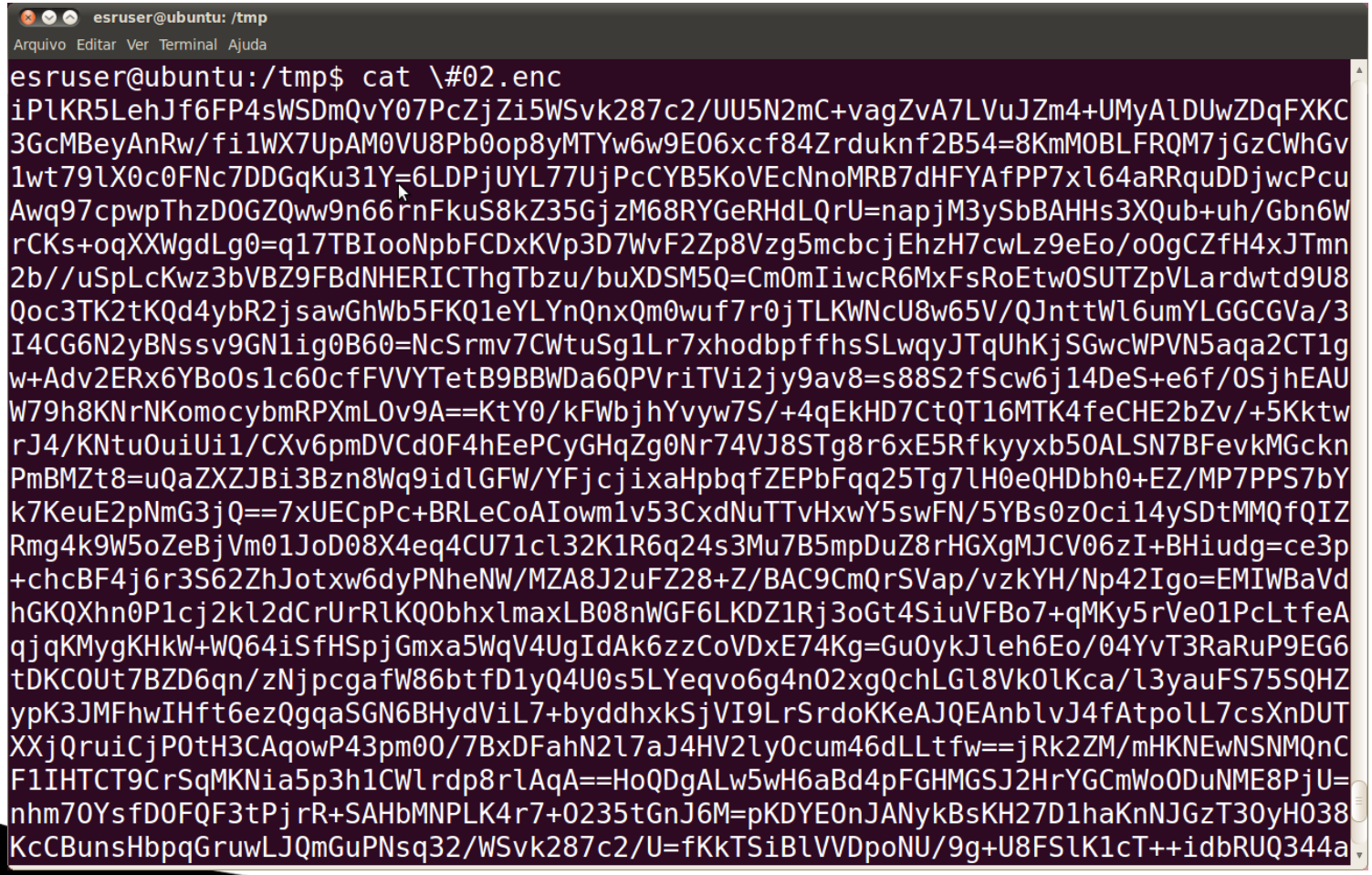


File #1 – Description of cipher

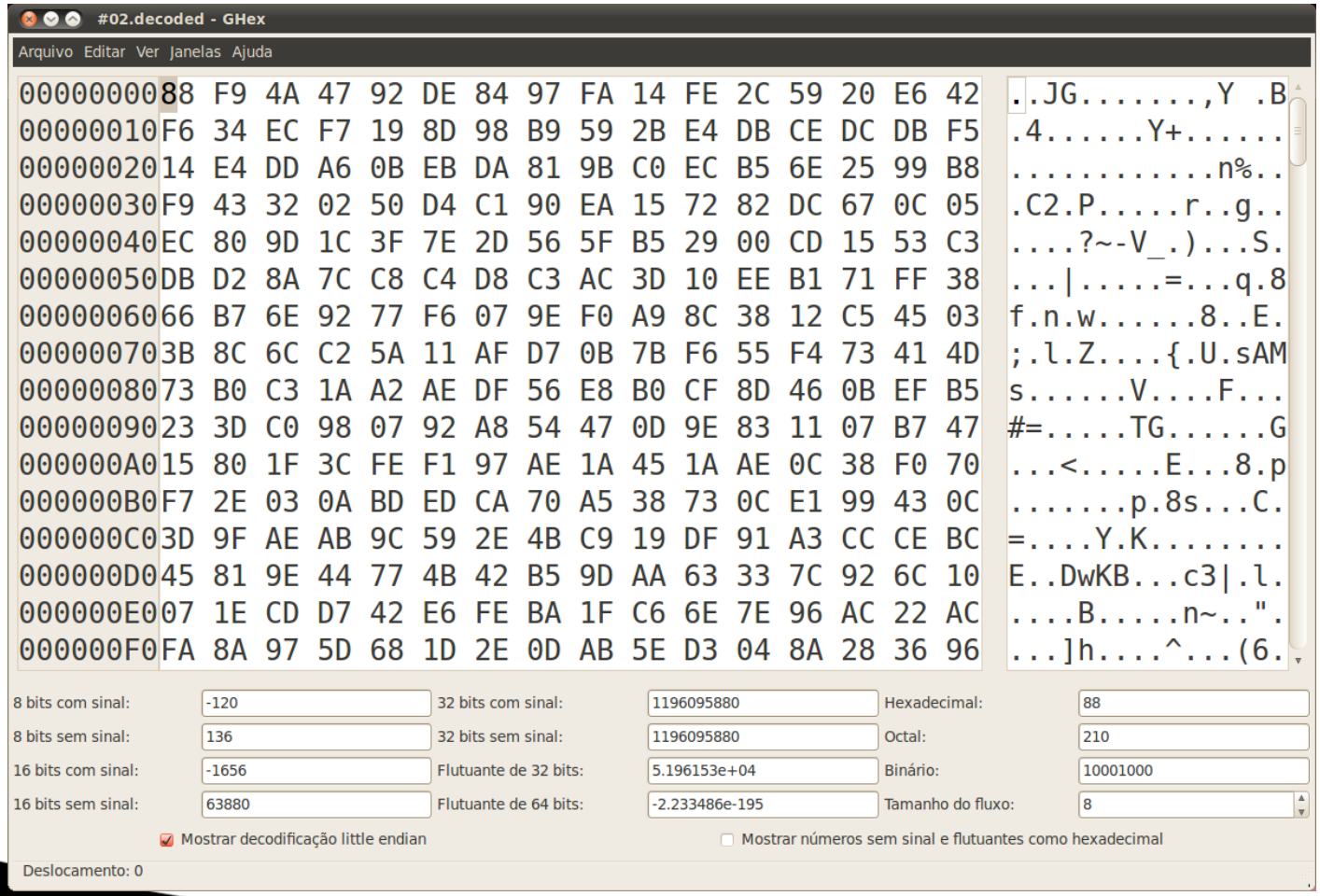
- **Alphabet of definition:** $\mathcal{A} = \{0, 1, 2, 3, \ldots, 255\}$
- **Plain text:** $M = m_0 m_1 m_2 \ldots m_{t-1}$, $m_i \in \mathcal{A}$
- **Cipher text:** $C = c_0 c_1 c_2 \ldots c_{t-1}$, $c_i \in \mathcal{A}$
- **Key:** $K = k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7 k_8$

  $= 0x52\ 4f\ 42\ 49\ 4e\ 20\ 48\ 4f\ 4f$
- **Encryption function:** $c_i = m_i + k_{(i \bmod 9)} \bmod 256$
- **Decryption function:** $m_i = c_i - k_{(i \bmod 9)} \bmod 256$

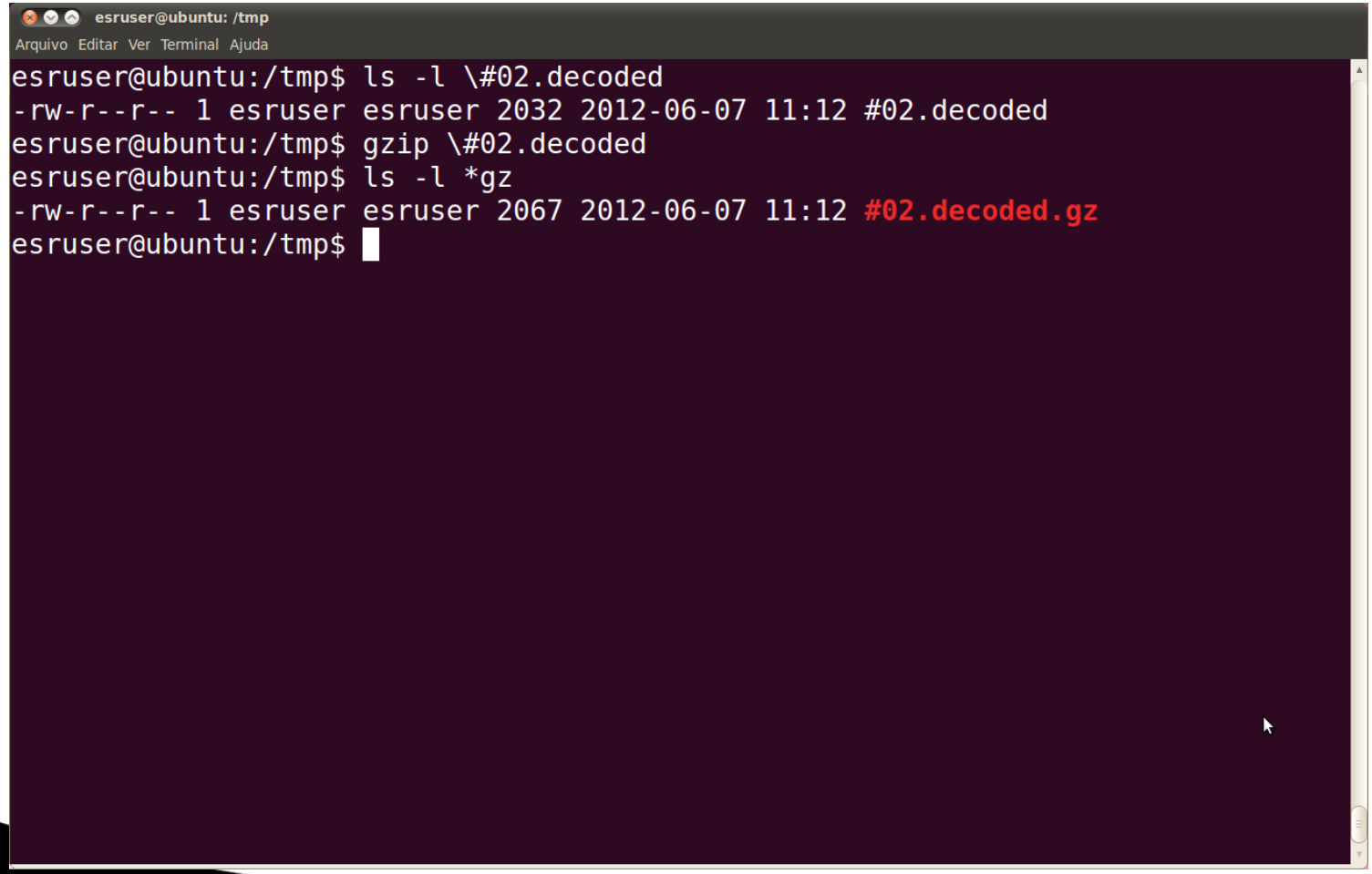
File #2 – Sample



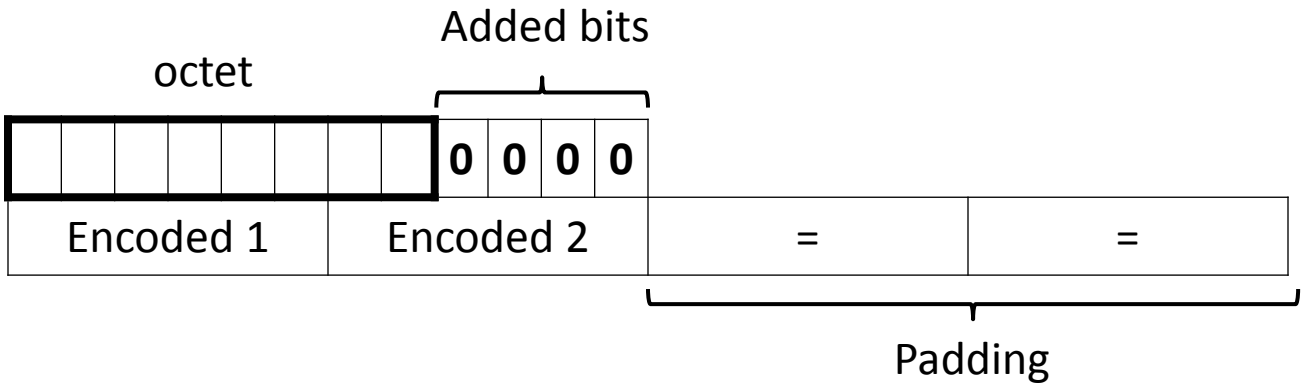
File #2 – Base64 decoded

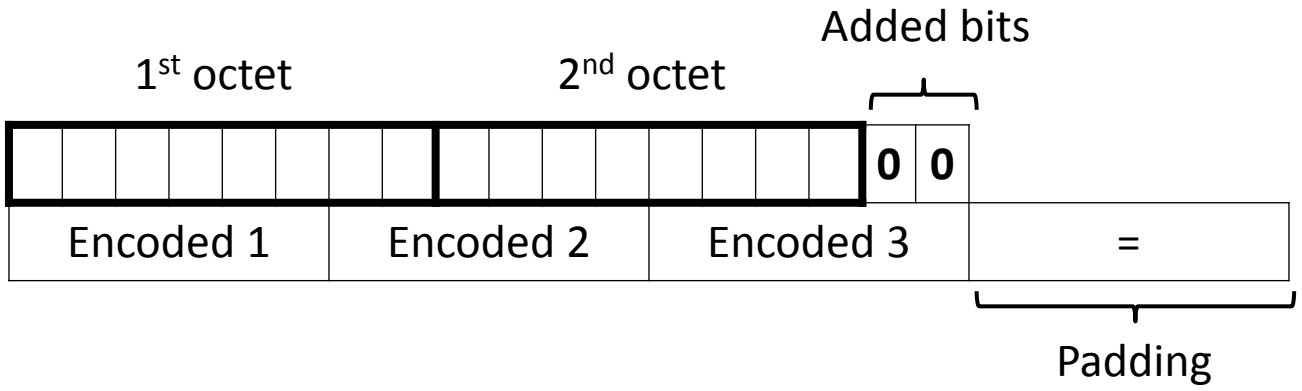


File #2 – Redundancy check



File #2 – Base64 review





File #2 – Block size?

1) Length = 56 Base64 chars.
2) Ends with "==".
3) Therefore input length equals 40 bytes.
4) Possible block size: 64 bits.

File #2 – Candidate ciphers

- DES.
- 2TDES.
- 3TDES.
- FEAL.
- IDEA.
- SAFER.
- RC5.
- LOKI.
- Blowfish.

File #2 – String search

```
esruser@ubuntu: /tmp
Arquivo  Editar  Ver  Terminal  Ajuda
esruser@ubuntu:/tmp$ strings Portsys.exe.malware | grep -i "encrypt"
esruser@ubuntu:/tmp$ strings Portsys.exe.malware | grep -i "crypto"
esruser@ubuntu:/tmp$ strings Portsys.exe.malware | grep -i "cipher"
ECipherException
LbCipher
esruser@ubuntu:/tmp$ strings Portsys.exe.malware | grep -i "DES"
IDesignerNotify
DesignSize
IDesignerHook,(A
poDesigned      poDefault
poDesktopCenter
        dmDesktop       dmPrimary
        OnDestroyT
GetDesktopWindow
DestroyWindow
DestroyMenu
DestroyIcon
DestroyCursor
ImageList_Destroy
esruser@ubuntu:/tmp$ strings Portsys.exe.malware | grep -i "bf"
esruser@ubuntu:/tmp$ strings Portsys.exe.malware | grep -i "blowfish"
esruser@ubuntu:/tmp$
```

File #2 – Narrowing the options

- LbCipher is a library for Delphi.
- It implements the following algorithms from our list:
  - Blowfish (ECB, CBC).
  - DES (ECB, CBC).
  - 2TDES (ECB, CBC).
  - 3TDES (ECB, CBC).

File #2 – Starting with DES

- DES is a 64-bit block cipher.
- The cipher employs a 64-bit key of which only 56 bits are effective.
- Based on a Feistel network.
- It is possible to search the entire key space using special purpose hardware[1], which was first built in 1998.

File #2 – Inside DES (1)

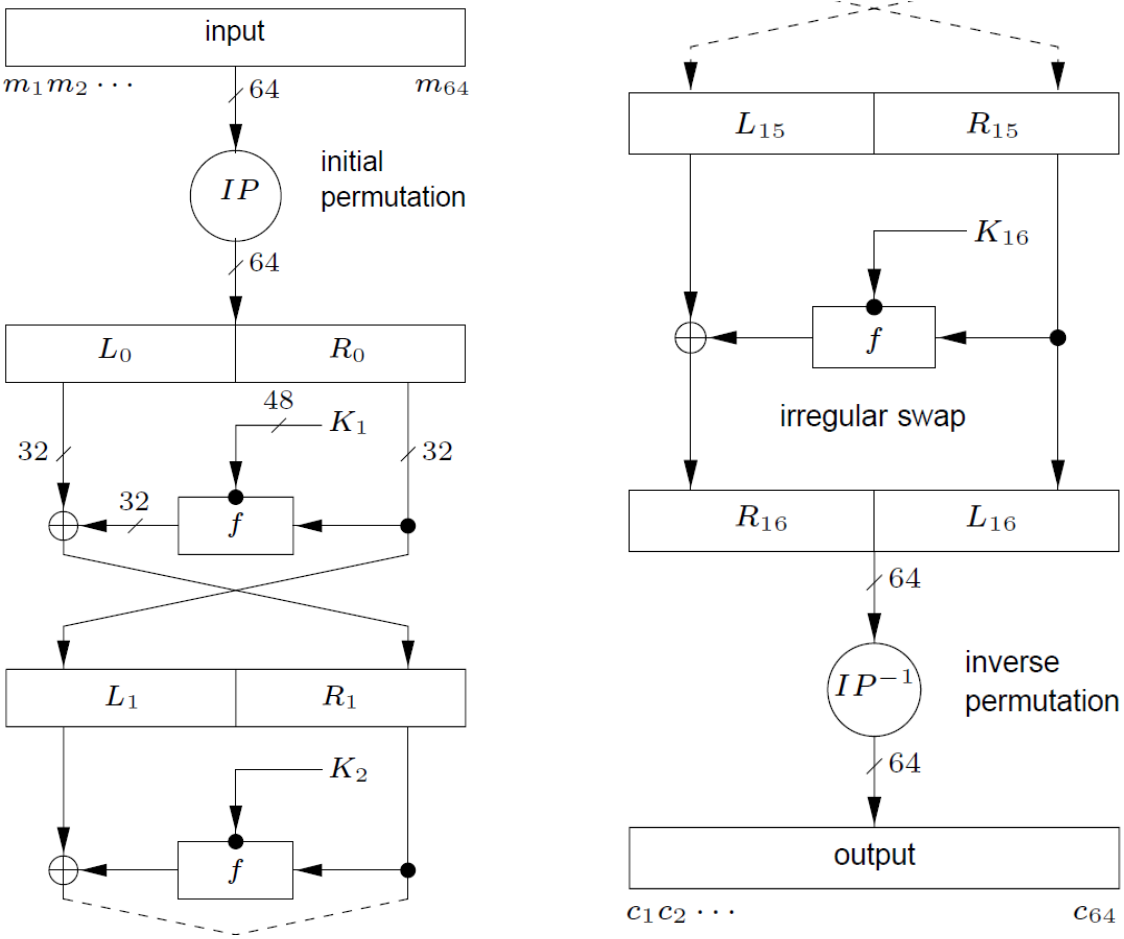


Figure: DES rounds.

Source: [2] HAC.

File #2 – Inside DES (2)

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Figure: DES initial permutation and inverse.

Source: [2] HAC.

File #2 – Inside DES (3)

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Figure: DES round function expansion E and permutation P.

Source: [2] HAC.

File #2 – Inside DES (4)

PC1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
above for C_i; below for D_i						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Figure: DES key schedule bit selections.

Source: [2] HAC.

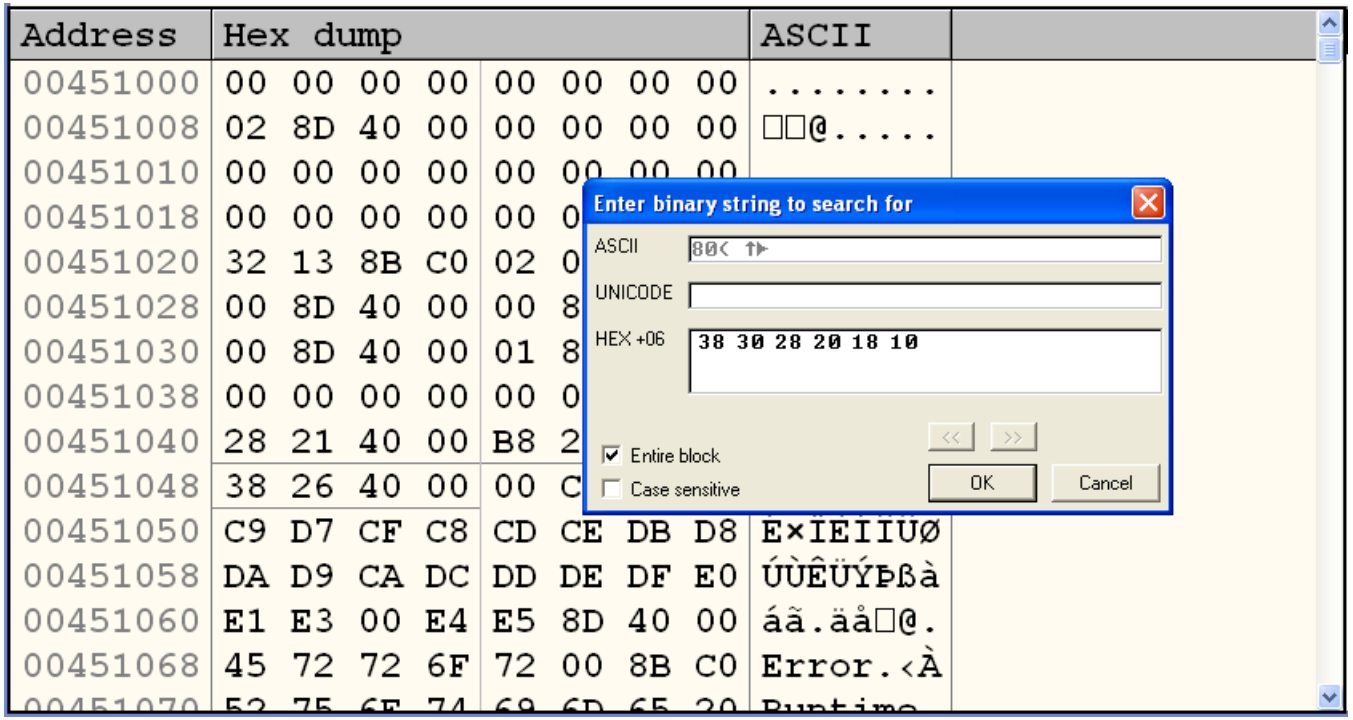
File #2 – From LbCipher

```
procedure InitEncryptDES(const Key : TKey64;
        var Context : TDESContext; Encrypt : Boolean);
const PC1 : array [0..55] of Byte = (56, 48, 40, 32, 24, 16, 8, 0,
57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51,
43, 35, 62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21, 13, 5,
60, 52, 44, 36, 28, 20, 12, 4, 27, 19, 11, 3);
PC2 : array [0..47] of Byte = (13, 16, 10, 23, 0, 4, 2, 27, 14, 5, 20,
9, 22, 18, 11, 3, 25, 7, 15, 6, 26, 19, 12, 1, 40, 51, 30, 36, 46, 54,
29, 39, 50, 44, 32, 47, 43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28,
31);
```

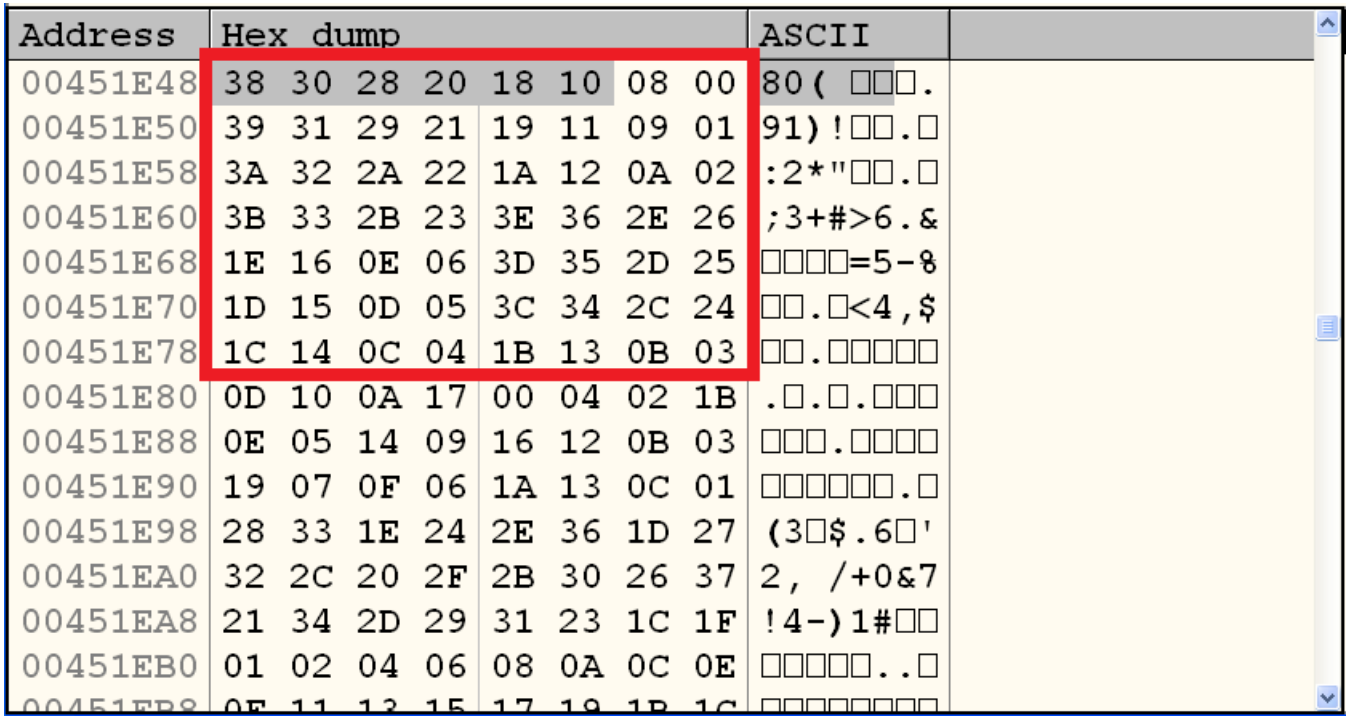
File #2 – Next steps

- Load the malware in OllyDbg.
- Search for PC1 and use it to locate the address of InitEncryptDES, if present.
- Set a breakpoint in that address.
- Run the malware.
- Extract the key from the first parameter.

File #2 – Finding PC1 (1)

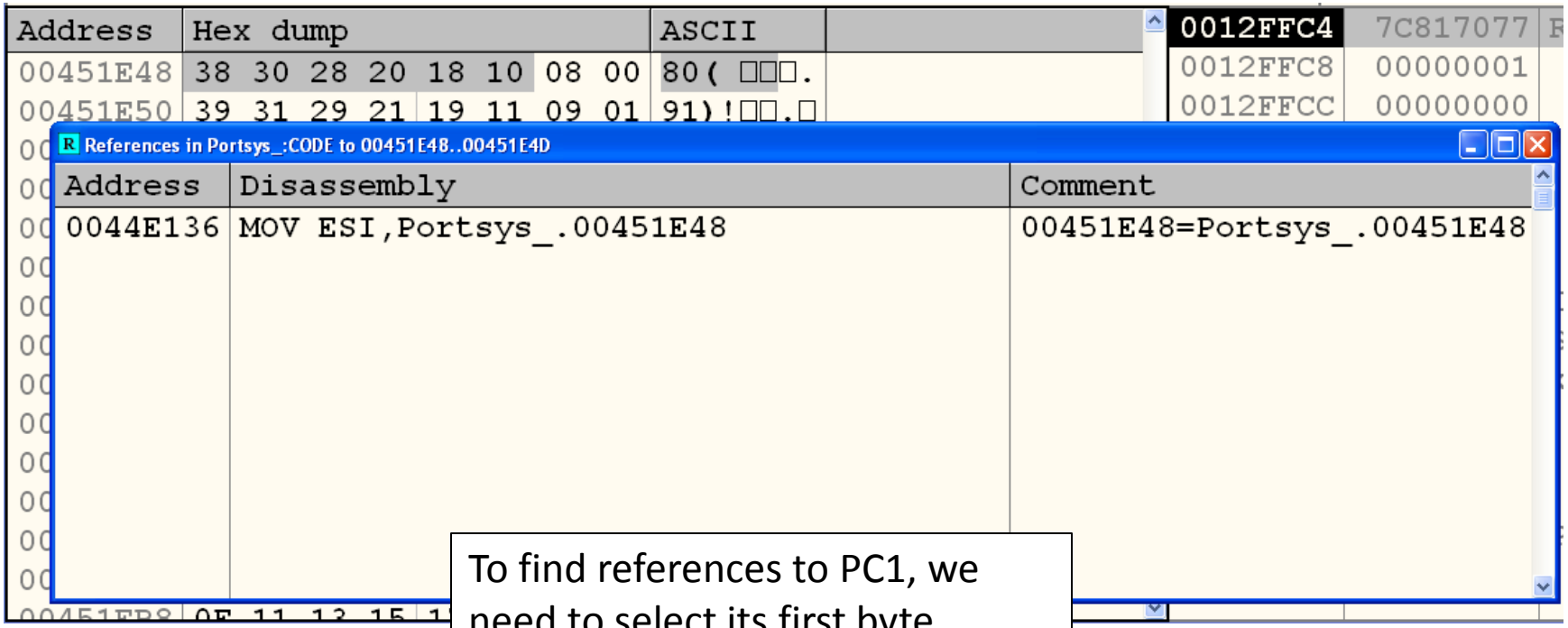


File #2 – Finding PC1 (2)



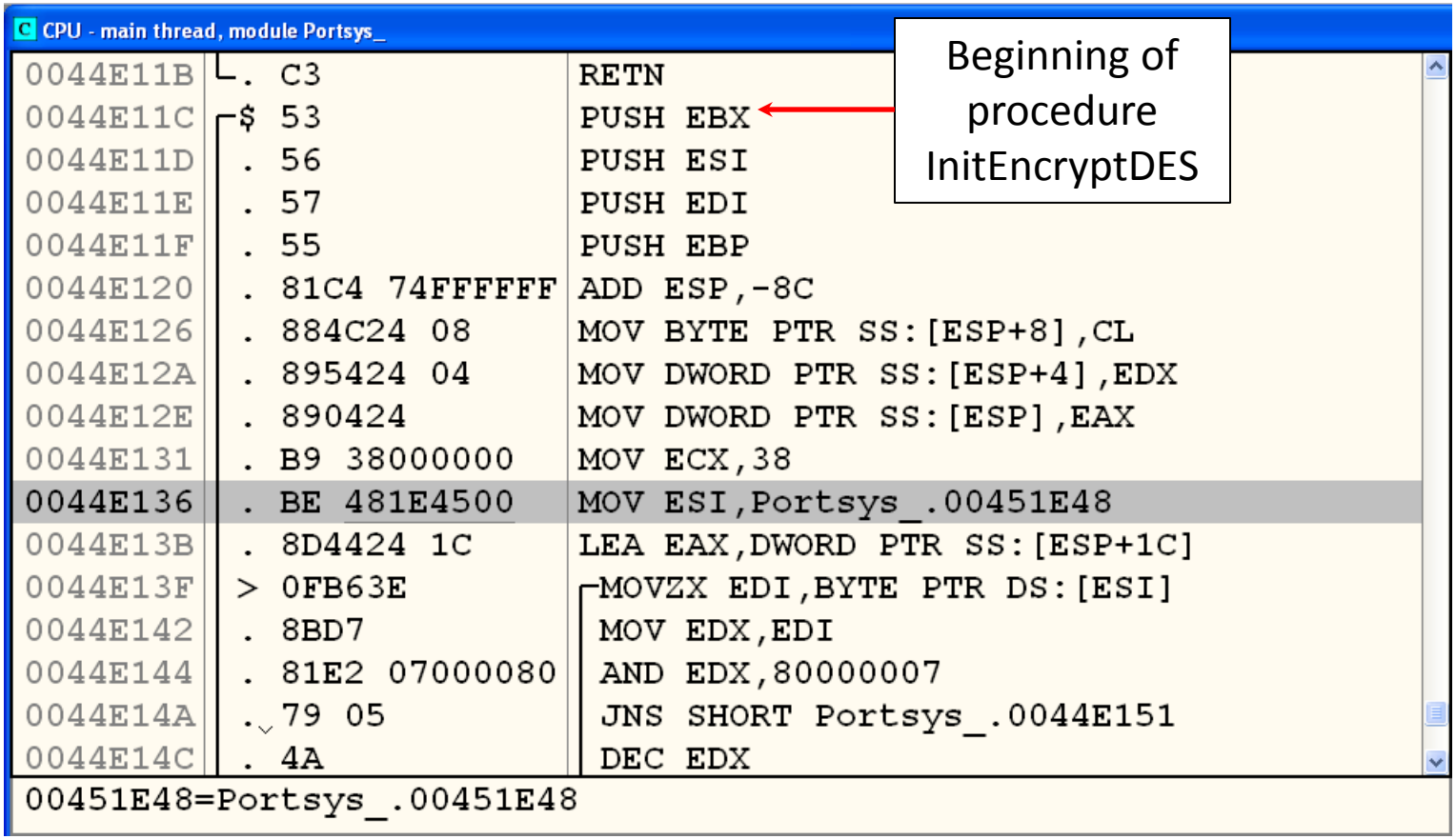
Address	Hex dump								ASCII
00451E48	38	30	28	20	18	10	08	00	80(□□□.
00451E50	39	31	29	21	19	11	09	01	91)!□□.□
00451E58	3A	32	2A	22	1A	12	0A	02	:2*"□□.□
00451E60	3B	33	2B	23	3E	36	2E	26	;3+#>6.&
00451E68	1E	16	0E	06	3D	35	2D	25	□□□□=5-%
00451E70	1D	15	0D	05	3C	34	2C	24	□□.□<4,$
00451E78	1C	14	0C	04	1B	13	0B	03	□□.□□□□□
00451E80	0D	10	0A	17	00	04	02	1B	.□.□.□□□
00451E88	0E	05	14	09	16	12	0B	03	□□□.□□□□
00451E90	19	07	0F	06	1A	13	0C	01	□□□□□□.□
00451E98	28	33	1E	24	2E	36	1D	27	(3□$.6□'
00451EA0	32	2C	20	2F	2B	30	26	37	2, /+0&7
00451EA8	21	34	2D	29	31	23	1C	1F	!4-)1#□□
00451EB0	01	02	04	06	08	0A	0C	0E	□□□□□..□
00451EB8	0F	11	13	15	17	19	1B	1C	□□□□□□□□

File #2 – References

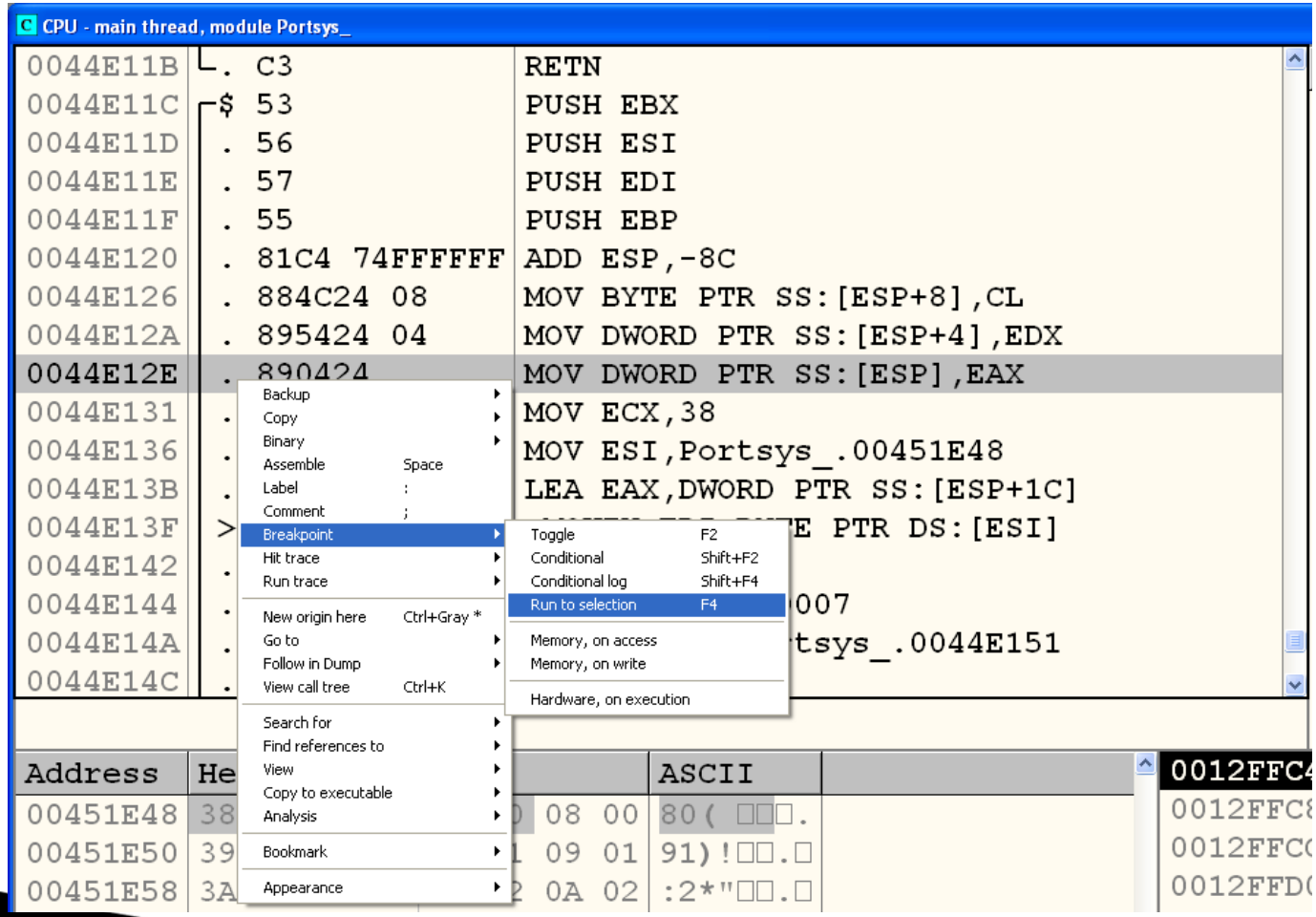


To find references to PC1, we need to select its first byte (0x38) and press Ctrl+R.

File #2 - Beginning of the function



File #2 – Running the malware



File #2 – Which parameter?

- Remember the procedure signature is as follows:

```
procedure InitEncryptDES(
    const Key : TKey64;
    var Context : TDESContext;
    Encrypt : Boolean);
```

- **TKey64** definition:

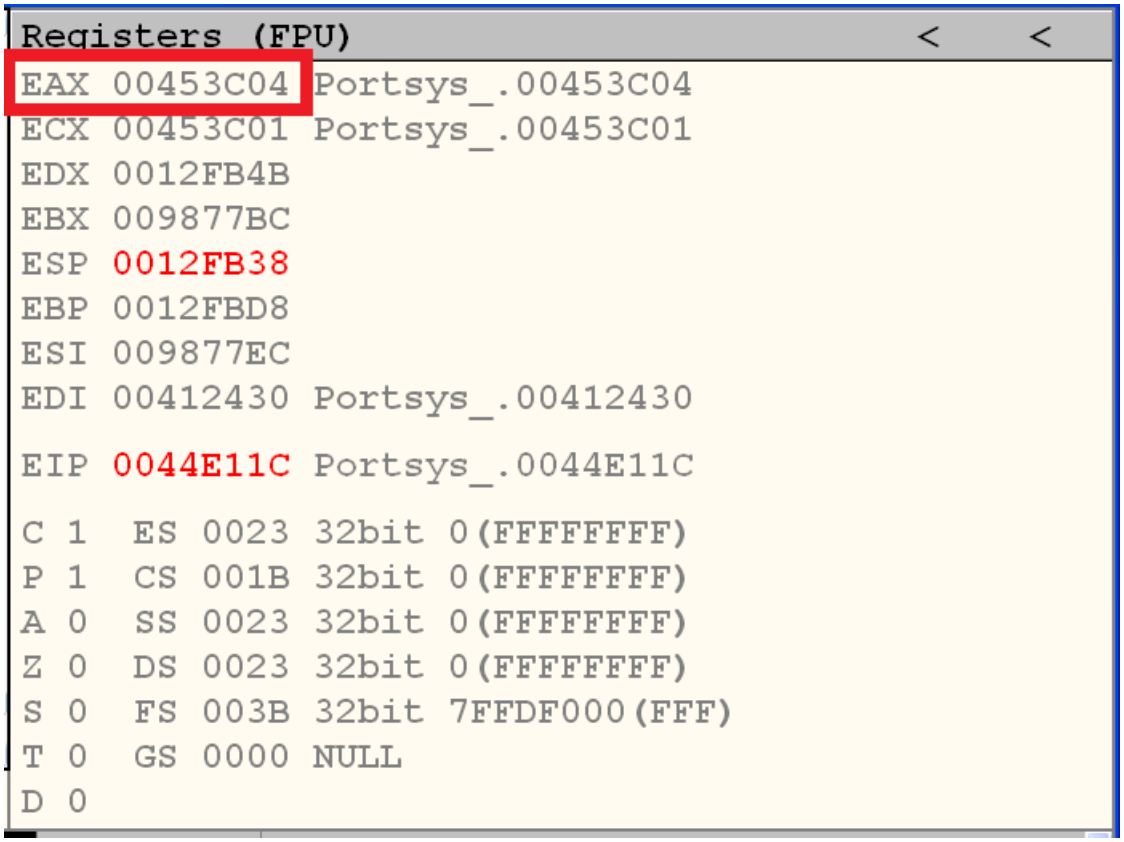
```
TKey64  = array [0..7] of Byte;
```

- A **TKey64** value can not be stored by a single register in a 32-bit architecture.

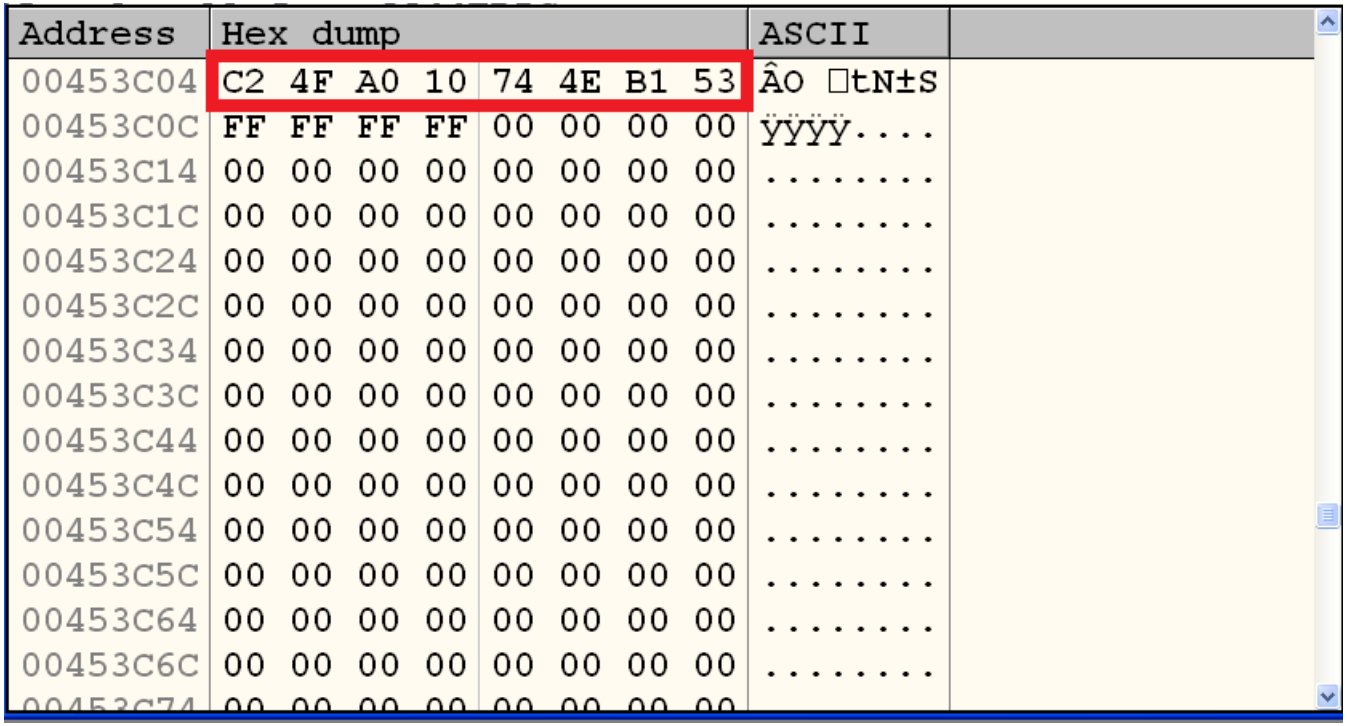
File #2 – Calling convention

- Delphi's calling convention (left-to-right):
  - 1st parameter: EAX.
  - 2nd parameter: EDX.
  - 3rd parameter: ECX.
  - Remaining parameters: stack.

File #2 – Key address



File #2 – Key value



Address	Hex dump								ASCII
00453C04	C2	4F	A0	10	74	4E	B1	53	ÂO □tN±S
00453C0C	FF	FF	FF	FF	00	00	00	00	ÿÿÿÿ....
00453C14	00	00	00	00	00	00	00	00	....
00453C1C	00	00	00	00	00	00	00	00	....
00453C24	00	00	00	00	00	00	00	00	....
00453C2C	00	00	00	00	00	00	00	00	....
00453C34	00	00	00	00	00	00	00	00	....
00453C3C	00	00	00	00	00	00	00	00	....
00453C44	00	00	00	00	00	00	00	00	....
00453C4C	00	00	00	00	00	00	00	00	....
00453C54	00	00	00	00	00	00	00	00	....
00453C5C	00	00	00	00	00	00	00	00	....
00453C64	00	00	00	00	00	00	00	00	....
00453C6C	00	00	00	00	00	00	00	00	....

File #2 – Description of cipher

- **Encryption algorithm:** DES.
- **Mode of operation:** ECB.
- **Key:** $K = 0xc24fa010744eb153$

Alternative for finding keys

- A properly generated key is entropic.
- Information, on the other hand, is structured.
- Based on those facts, in 1999, Shamir and Someren[3] proposed a way of finding stored keys.
- The basic idea is to traverse memory and identify the region with more entropy.
- One way of doing that is to set a window size and count the number of different elements on each window.

File #3 – Sample

50E96823#0851CDA207333E24 1.0.6 St - P: 6 R: 11

CFT:1.0.2

PA: 3

C3@158BF7627CD2750FF53D7288C863F7C7041221CD8E77B6A7F7833815075091A23EB3ADA
2352ADFE9514952DE6DF8B619D41E51DFB7C0196A104F994920E2434716699DEF0DA48E624
CEC0953F7BE159E0B43F3862C4A8D8FE1476F7939F72F99A049CAC2DC1DE0E6BB91066FF3E9
20283A373E8B94DF3D39F06FCB6A29B9E5DCF20A0D02DE8F288F5C2737D1D64E1E25AA51A
42C0AAE3ABFE354EBCE781342A6D84413391F4038EDB213AA87870D25FC06DD05DBF3EEB6
84665A7E20C080F196BA42D96CFE0FA08FF64FF9B3C08CA3765768EDCBEDF620562ADB442C
6A1191A1A137E50C7F75C629AEB702F09F81107

PF: 3

50E96832#K@881A6DC9E4470F

50E96837#K@06BB

50E9683C#K@3FE759EE

File #3 – Description of cipher

- **Alphabet of definition:** $\mathcal{A} = \{0, 1, 2, 3, \ldots, 255\}$
- **Plain text:** $M = m_0 m_1 m_2 \ldots m_{t-1}$, $m_i \in \mathcal{A}$
- **Cipher text:** $C = c_0 c_1 c_2 \ldots c_{t-1}$, $c_i \in \mathcal{A}$
- **Key:** $K = k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7 k_8 k_9 k_{10}$
- **Encryption function:** $c_i = m_i + k_{(i \bmod 11)} \bmod 256$
- **Decryption function:** $m_i = c_i - k_{(i \bmod 11)} \bmod 256$

References

- [1] Electronic Frontier Foundation, *Cracking Des: Secrets of Encryption Research, Wiretap Politics & Chip Design*, O'Reilly Media, 1998.
- [2] Menezes, A., van Oorschot, P, and Vanstone, S., *Handbook of Applied Cryptography*, CRC Press, 2001.
- [3] Shamir, A. and van Someren,N., *Playing "Hide and Seek" with Stored Keys*, in FC'99 Proc. of the 3rd Intl. Conference on Financial Cryptography, 1999.

Thank you for listening!

Questions?

Nelson Uto

uto@cpqd.com.br