MRMCD14

SSBAUSTELLE IT-SECURITY GROSSBAUSTELLE IT

Ange Albertini

2014/09/05

ANGE ALBERTINI

reverse engineering

VISUAL DOCUMENTATIONS

@angealbertini

ange@corkami.com

http://www.corkami.com

an *encrypted* file is
not always *encrypted*

encrypt(file)* is not always *random

Let’s take an example: this is a JPEG picture...

AES( )

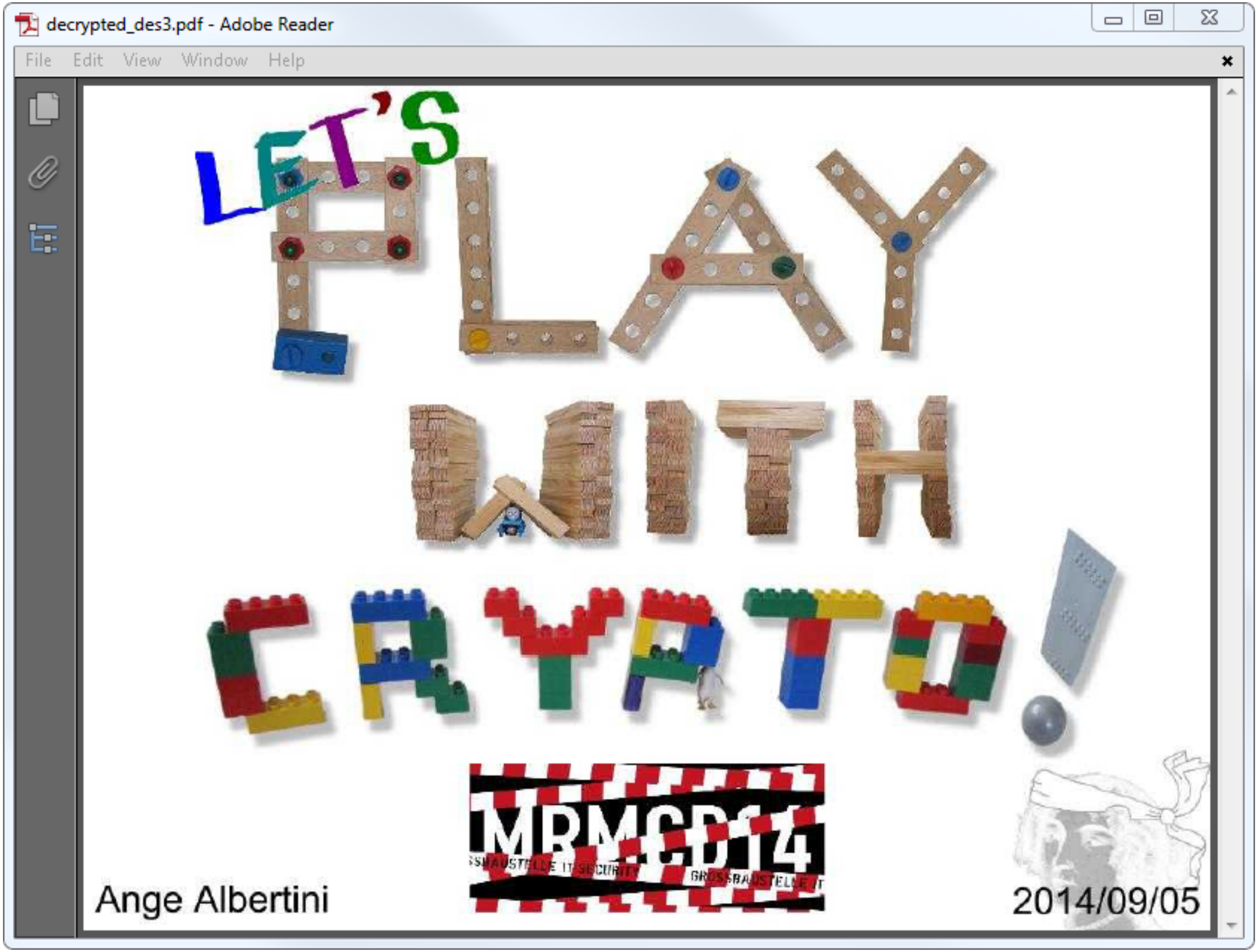
...if you encrypt it with AES...

MRMCD14
SSBAUSTELLE IT-SECURITY
GROSSBAUSTELLE IT

… you get this PNG picture...

3DES( )

...and if you ***de***crypt it with Triple DES...



Ange Albertini

2014/09/05

...you get a PDF document.

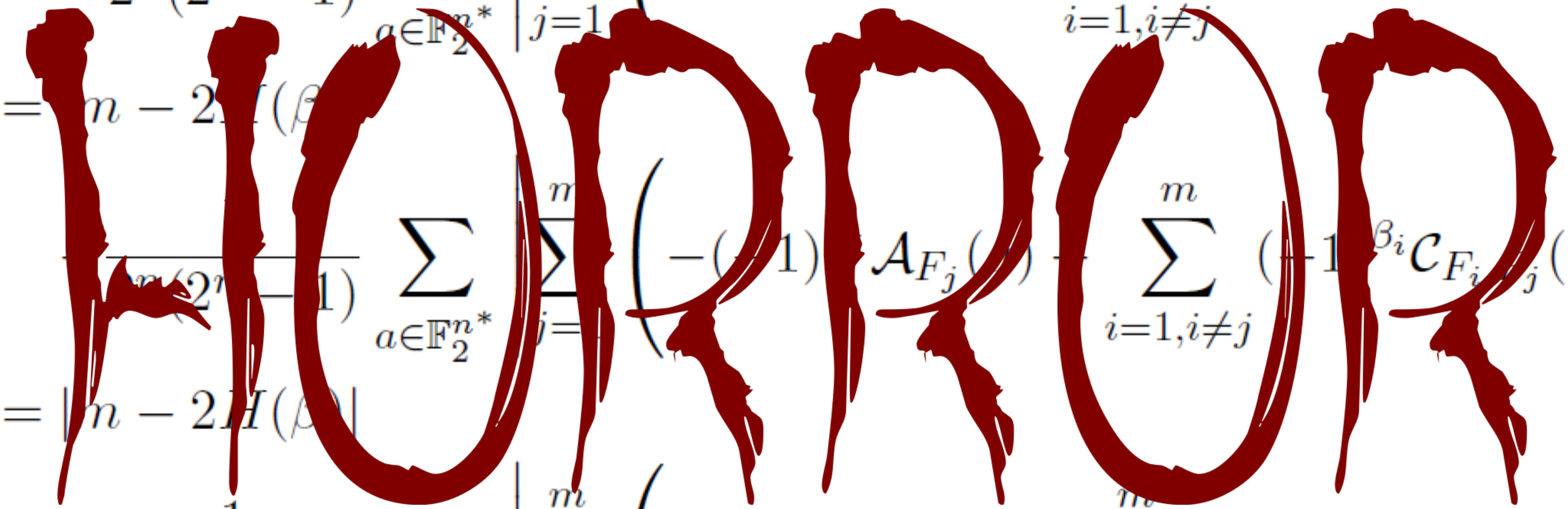
Don’t worry!

I’ll keep it simple...

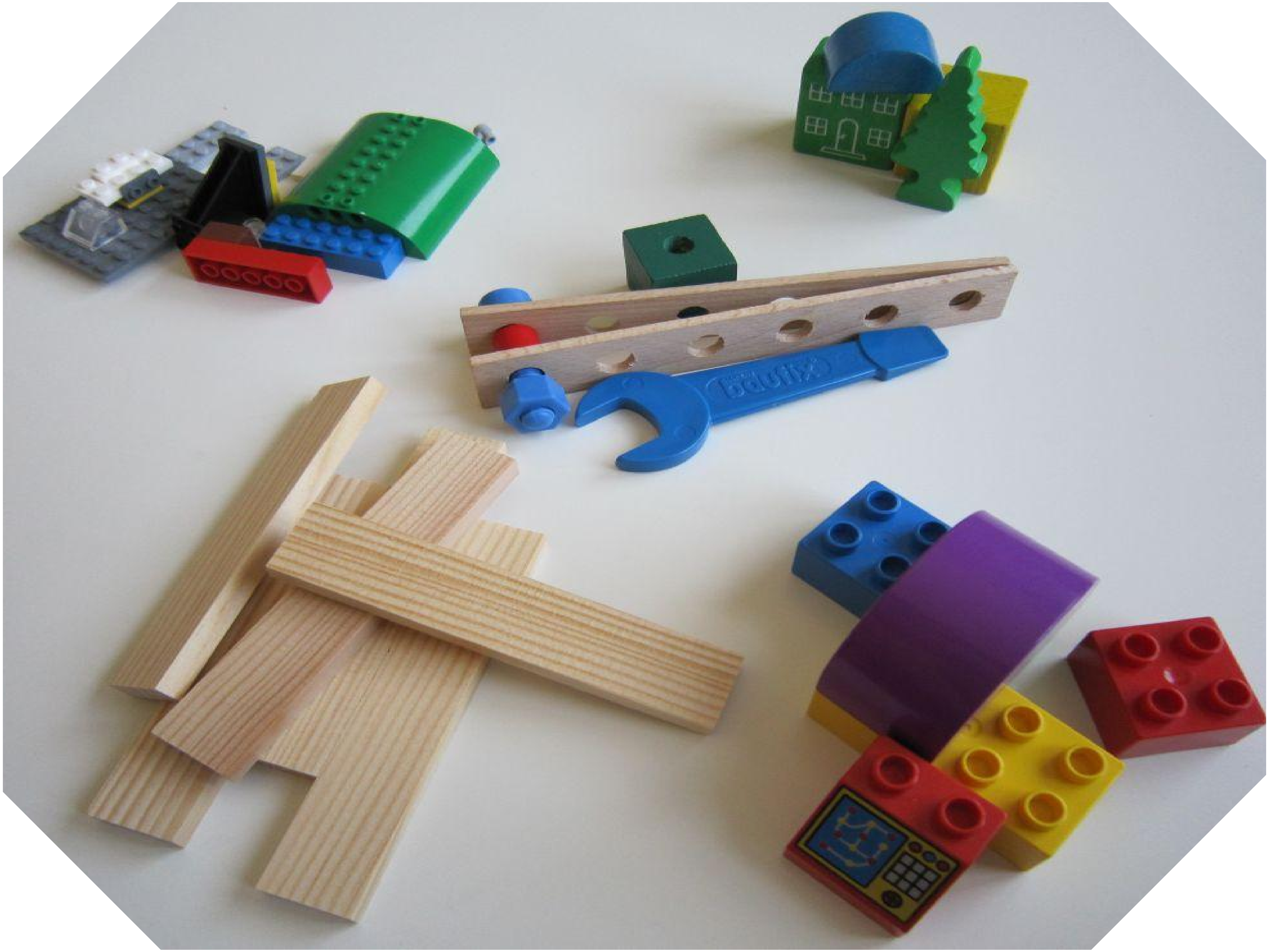
$$
\begin{aligned}
\tau'^{\overline{\beta}}_F &= |m - 2H(\overline{\beta})| \\
&\quad - \frac{1}{2^n(2^n-1)} \sum_{a \in \mathbb{F}_2^{n*}} \left| \sum_{j=1}^{m} \left((-1)^{\overline{\beta}_j} \mathcal{A}_{F_j}(a) + \sum_{i=1, i \neq j}^{m} (-1)^{\overline{\beta}_i} \mathcal{C}_{F_i,F_j}(a) \right) \right| \\
&= |m - 2H(\beta)| \\
&\quad - \frac{1}{2^n(2^n-1)} \sum_{a \in \mathbb{F}_2^{n*}} \left| \sum_{j=1}^{m} \left(-(-1)^{\beta_j} \mathcal{A}_{F_j}(a) - \sum_{i=1, i \neq j}^{m} (-1)^{\beta_i} \mathcal{C}_{F_i,F_j}(a) \right) \right| \\
&= |m - 2H(\beta)| \\
&\quad - \frac{1}{2^n(2^n-1)} \sum_{a \in \mathbb{F}_2^{n*}} \left| \sum_{j=1}^{m} \left((-1)^{\beta_j} \mathcal{A}_{F_j}(a) + \sum_{i=1, i \neq j}^{m} (-1)^{\beta_i} \mathcal{C}_{F_i,F_j}(a) \right) \right| \\
&= \tau'^{\beta}_F .
\end{aligned}
$$

...because crypto is (too) hard!

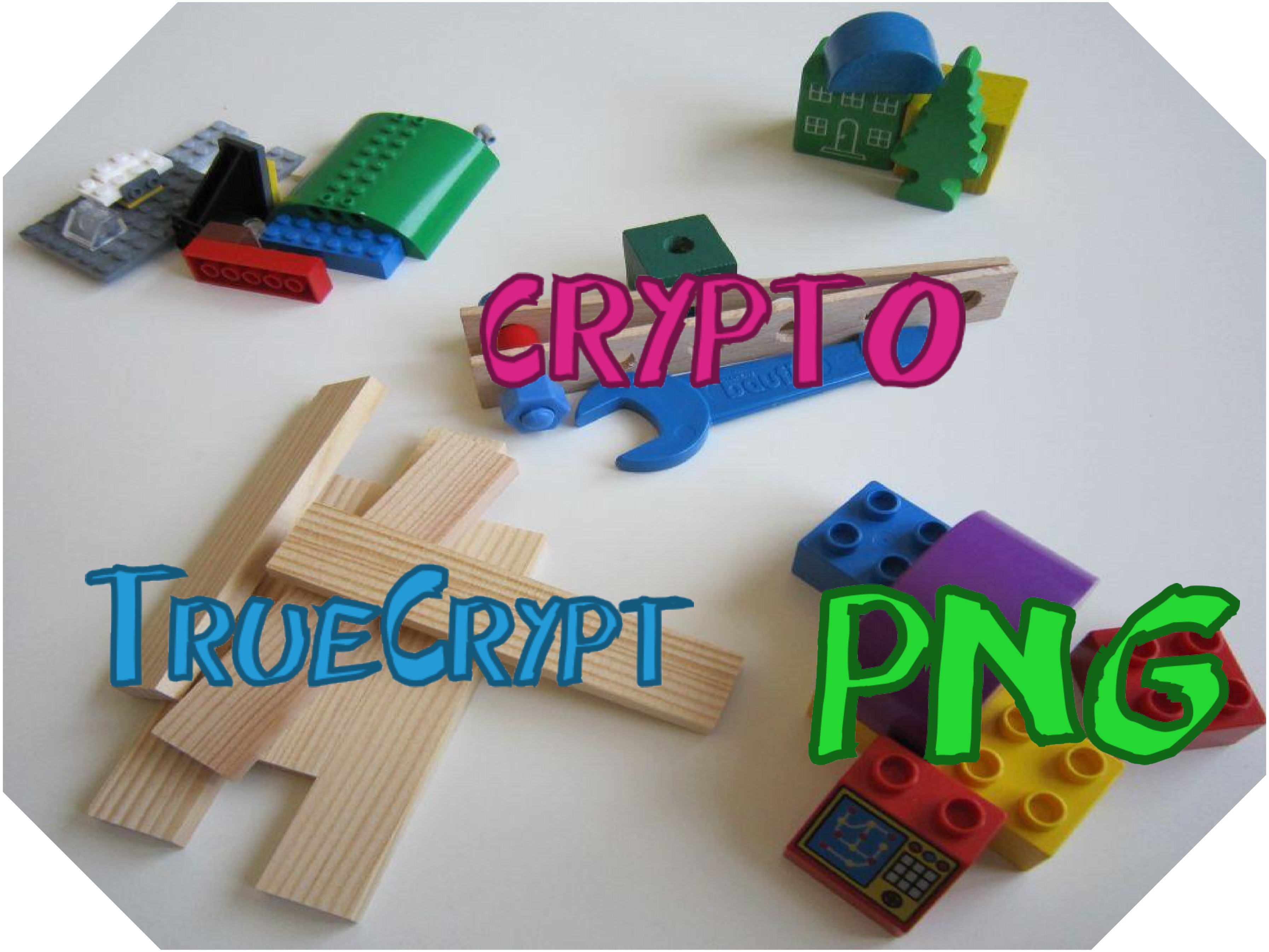
$$
\begin{aligned}
\tau'^{\overline{\beta}}_{F} &= |m - 2H(\overline{\beta})| \\
&\quad - \frac{1}{2^n(2^n-1)} \sum_{a\in\mathbb{F}_2^{n*}} \left| \sum_{j=1}^{m} \left((-1)^{\overline{\beta}_j} \mathcal{A}_{F_j}(a) + \sum_{i=1, i\neq j}^{m} (-1)^{\overline{\beta}_i} \mathcal{C}_{F_i,F_j}(a) \right) \right| \\
&= |m - 2H(\beta)| \\
&\quad - \frac{1}{2^n(2^n-1)} \sum_{a\in\mathbb{F}_2^{n*}} \left| \sum_{j=1}^{m} \left(-(-1)^{\beta_j} \mathcal{A}_{F_j}(a) - \sum_{i=1, i\neq j}^{m} (-1)^{\beta_i} \mathcal{C}_{F_i,F_j}(a) \right) \right| \\
&= |m - 2H(\beta)| \\
&\quad - \frac{1}{2^n(2^n-1)} \sum_{a\in\mathbb{F}_2^{n*}} \left| \sum_{j=1}^{m} \left((-1)^{\beta_j} \mathcal{A}_{F_j}(a) + \sum_{i=1, i\neq j}^{m} (-1)^{\beta_i} \mathcal{C}_{F_i,F_j}(a) \right) \right| \\
&= \tau'^{\beta}_{F}.
\end{aligned}
$$



And this is my usual reaction...



...but I can still have fun with it...



...so let's play together !

AES

Advanced Encryption Standard

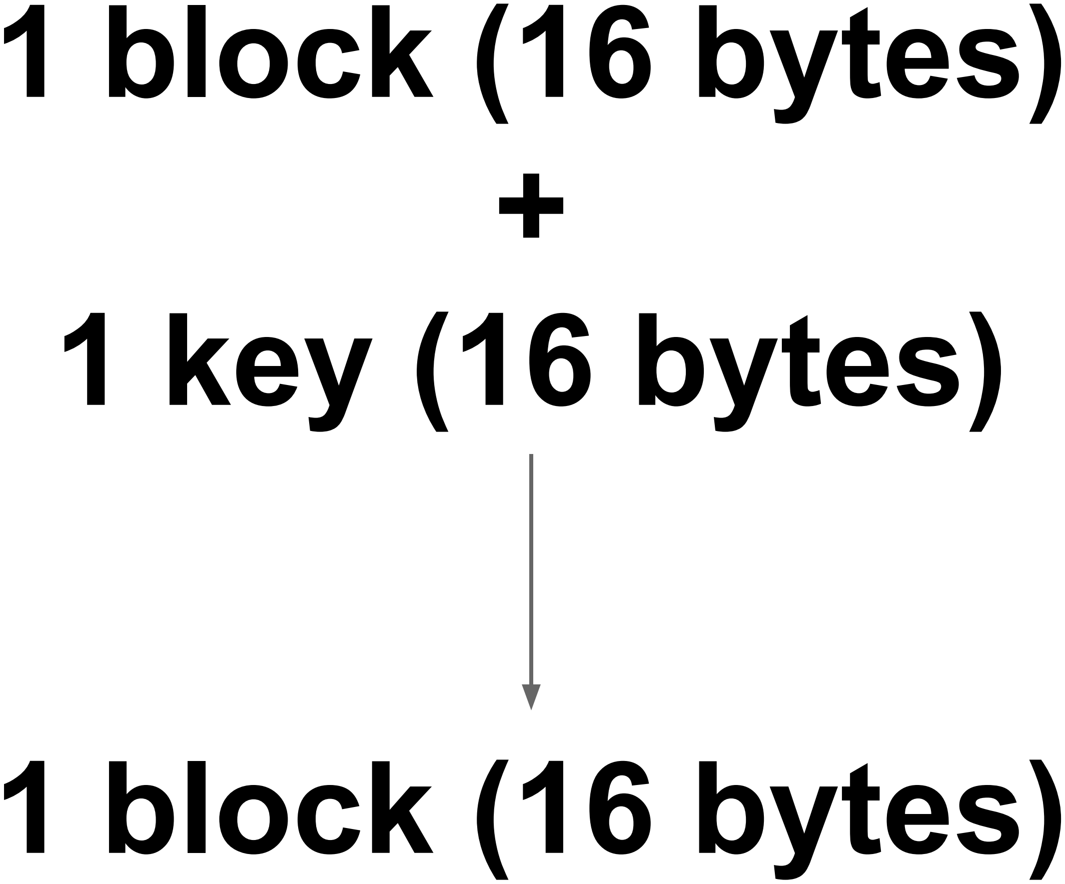
1 block (16 bytes)

1 block (16 bytes)

+

1 key (16 bytes)[*]

[*] in the case of AES-128,
so from now on, we'll say AES for AES-128.



1 block (16 bytes)

+

1 key (16 bytes)

1 block (16 bytes)

a block of text.

+

MySecretKey12345

↓

┐◄n╩i▐☼←∞└╞∙iû╨►

(BF 11 6E CA 69 DE 0F 1B EC C0 C6 F9 69 96 D0 10)

a block of text.

+

MySecretKey12346

↓

gO┼╗ÑëΩcë ▼LÇk╨î

(67 4F C5 BB A5 89 EA 63 89 20 1F 4C 80 6B D0 8C)

a block of text!

+

MySecretKey12345

↓

wε╩▬▄y&↕ú@αùαφ♣O

(77 EE CA 16 DC 79 26 12 A3 40 E0 97 E0 ED 05 4F)

Any change
in the key or input block
gives a completely
different output

we can’t
control the output

the differences are unpredictable

the opposite operation

a block of text.

+

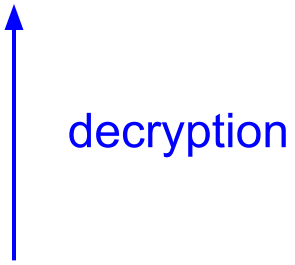
MySecretKey12345

encryption

┐◄n╩i▐☼←∞└╞∙iû╨►

(BF 11 6E CA 69 DE 0F 1B EC C0 C6 F9 69 96 D0 10)

a block of text.



decryption

MySecretKey12345

+

┐◄n╩i▐☼←∞└╞∙iû╨►

(BF 11 6E CA 69 DE 0F 1B EC C0 C6 F9 69 96 D0 10)

π╔6I►♣♫Σ♣╝╤→√çφ╡

(E3 C9 36 49 10 05 0E E4 05 BC D1 1A FB 87 ED B5)

decryption

MySecretKey12346

+

┐◄n╩i▐☼←∞└╞∙iû╨►

(BF 11 6E CA 69 DE 0F 1B EC C0 C6 F9 69 96 D0 10)

***with* the encryption key,**
we can restore
the original block

***without* the encryption key,**
we can’t do anything
with the encrypted block

“plaintext” and “crypted” are just names

encryption ⇔ decryption
are just inverse functions

a block of text.

+

MySecretKey12345

encryption

┐◄n╩i▐☼←∞└╞∙iû╨►

(BF 11 6E CA 69 DE 0F 1B EC C0 C6 F9 69 96 D0 10)

a block of text.

+

MySecretKey12345

↓ decryption

ä/ë-╦7 ↓h│☻⌂µ[←Ñ

(84 2F 89 2D CB 37 00 19 68 B3 02 7F E6 5B 1B A5)

a block of text.

encryption

MySecretKey12345

+

ä/ë-╦7 ↓h│☻⌂µ[←Ñ

(84 2F 89 2D CB 37 00 19 68 B3 02 7F E6 5B 1B A5)

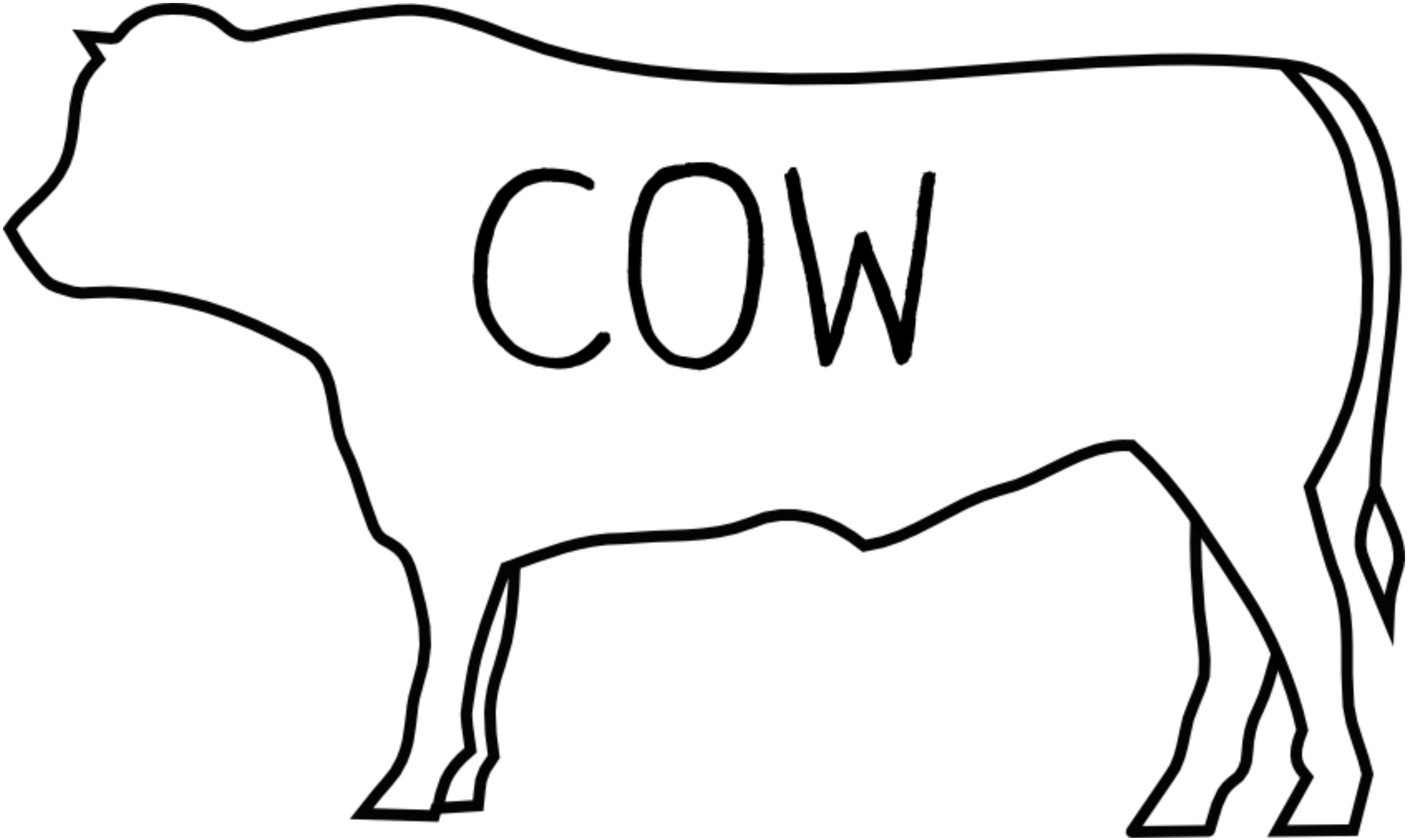
we *can* decrypt plaintext

we recover the original block via encryption
⇒ we can control encryption output

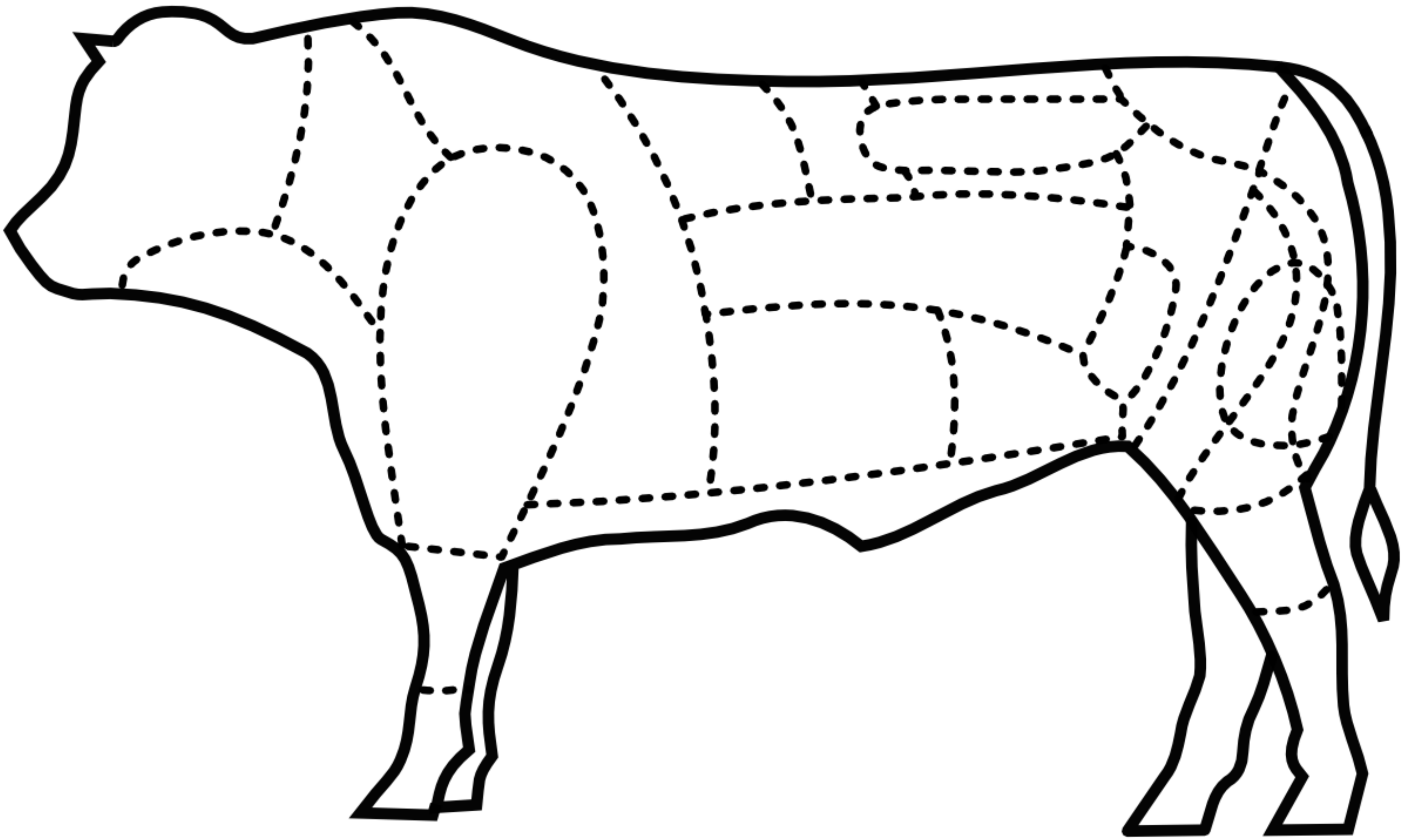
Recap

- AES encrypts a block
  - we don’t control the output
- an encrypted block can be restored
  - with the encryption key
- encryption ⇔ decryption are just inverse functions
  - we can decrypt plaintext
  - we can recover the original block via encryption
- we can’t control both input *and* output
  - one, or the other

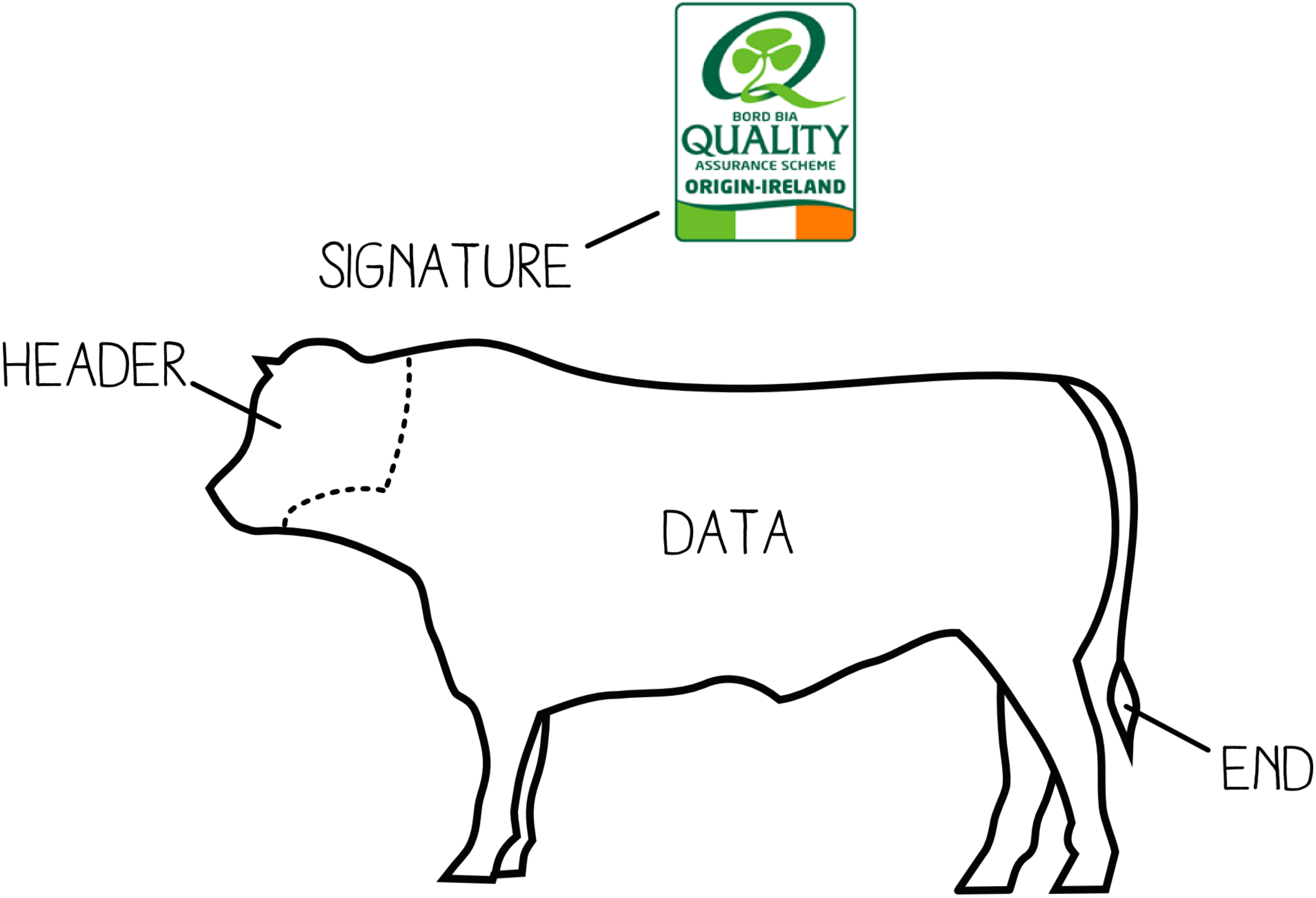
Now, let’s talk about…



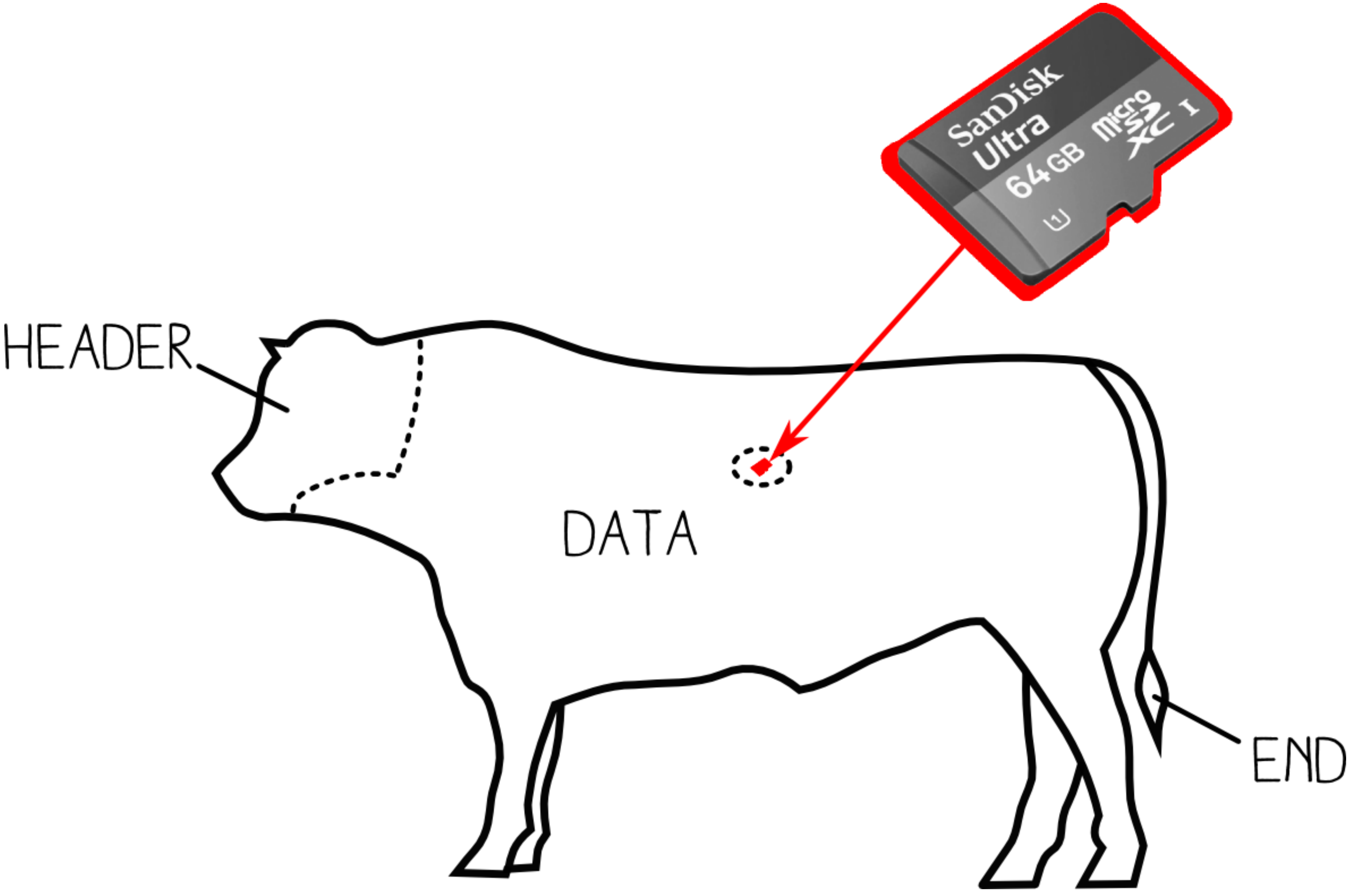
This is a cow. This is how a users see it.



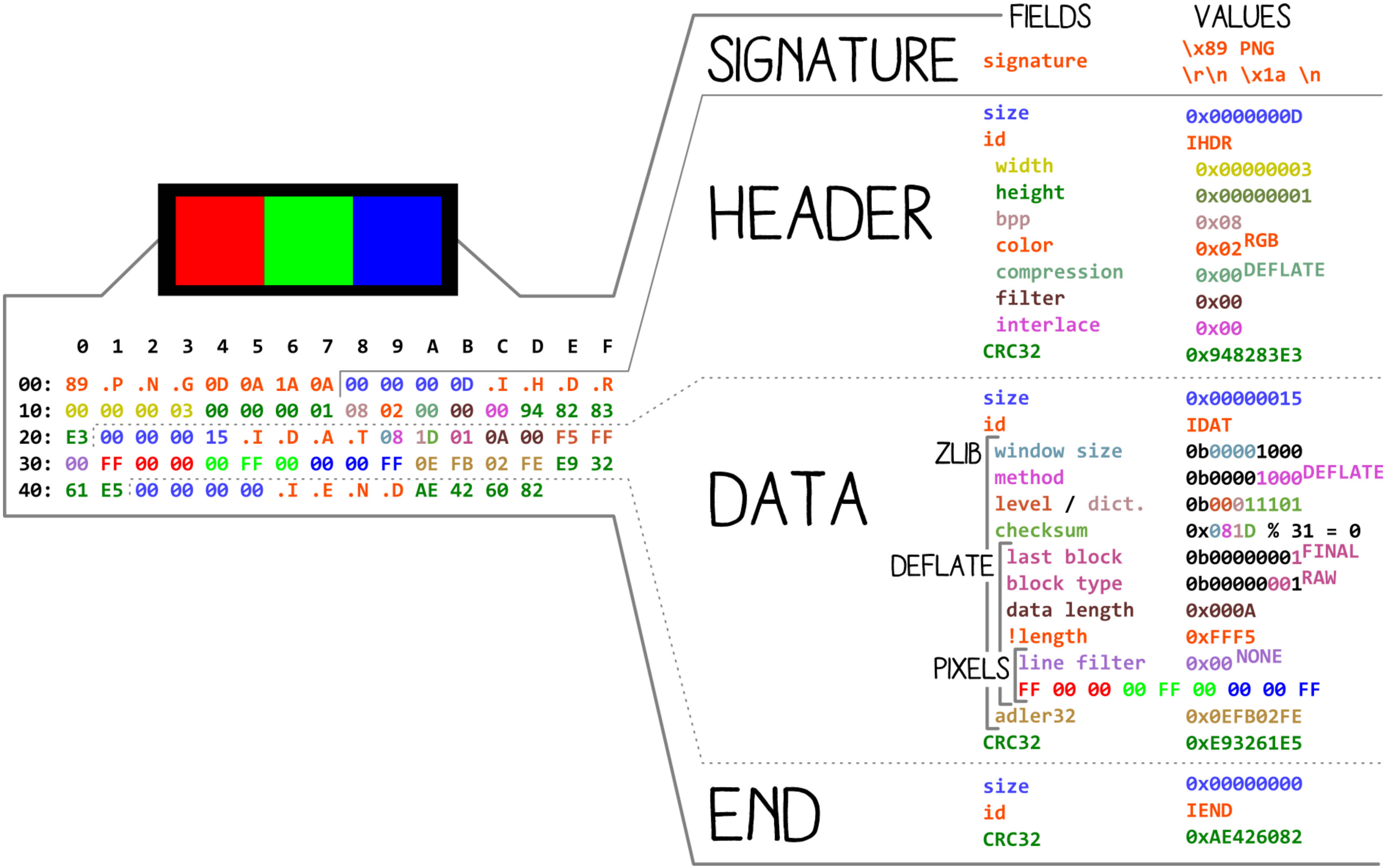
This is how an expert sees a cow.
It’s still the same cow, but it has some internal structure.



Let’s work with a simplified structure of beef chunks.



If our cow swallows a microSD, it’s still a valid cow!
Even if it contains foreign data, that is tolerated by the system.



```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 89 .P .N .G 0D 0A 1A 0A 00 00 00 0D .I .H .D .R
10: 00 00 00 03 00 00 00 01 08 02 00 00 00 94 82 83
20: E3 00 00 00 15 .I .D .A .T 08 1D 01 0A 00 F5 FF
30: 00 FF 00 00 00 FF 00 00 00 FF 0E FB 02 FE E9 32
40: 61 E5 00 00 00 00 .I .E .N .D AE 42 60 82
```

		FIELDS	VALUES
SIGNATURE		signature	\x89 PNG \r\n \x1a \n
HEADER		size	0x0000000D
		id	IHDR
		width	0x00000003
		height	0x00000001
		bpp	0x08
		color	0x02 RGB
		compression	0x00 DEFLATE
		filter	0x00
		interlace	0x00
		CRC32	0x948283E3
DATA		size	0x00000015
		id	IDAT
	ZLIB	window size	0b00001000
		method	0b00001000 DEFLATE
		level / dict.	0b00011101
		checksum	0x081D % 31 = 0
	DEFLATE	last block	0b00000001 FINAL
		block type	0b00000001 RAW
		data length	0x000A
		!length	0xFFF5
	PIXELS	line filter	0x00 NONE
		FF 00 00 00 FF 00 00 00 FF	
		adler32	0x0EFB02FE
		CRC32	0xE93261E5
END		size	0x00000000
		id	IEND
		CRC32	0xAE426082

Now, let’s look at the ***P***ortable ***N***etwork ***G***raphics format.

Chunk

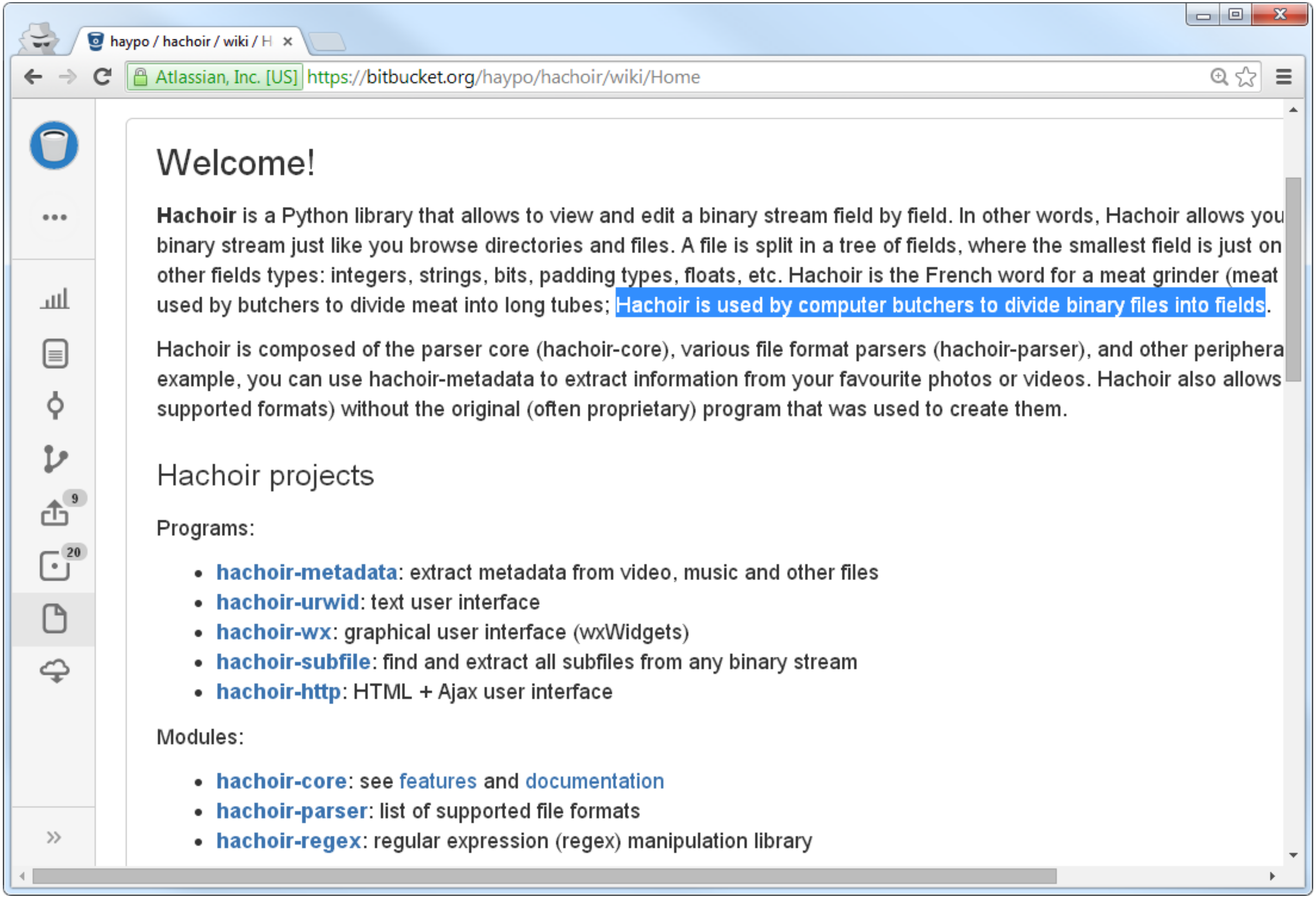
- The format is made of variable-sized pieces
  - critical or ancillary
- Common high-level structure
  - independent of the content and its interpretation

⇒ Store proprietary information while guaranteeing a minimal compatibility

https://www.google.com/images/srpr/logo11w.png

SHA-1 349841408d1aa1f5a8892686fbdf54777afc0b2c

Let’s take a real example, that you may have seen before.



haypo / hachoir / wiki / H
Atlassian, Inc. [US]
https://bitbucket.org/haypo/hachoir/wiki/Home
9
20

Welcome!

Hachoir is a Python library that allows to view and edit a binary stream field by field. In other words, Hachoir allows you binary stream just like you browse directories and files. A file is split in a tree of fields, where the smallest field is just on other fields types: integers, strings, bits, padding types, floats, etc. Hachoir is the French word for a meat grinder (meat used by butchers to divide meat into long tubes; Hachoir is used by computer butchers to divide binary files into fields.

Hachoir is composed of the parser core (hachoir-core), various file format parsers (hachoir-parser), and other periphera example, you can use hachoir-metadata to extract information from your favourite photos or videos. Hachoir also allows supported formats) without the original (often proprietary) program that was used to create them.

Hachoir projects

Programs:

- **hachoir-metadata**: extract metadata from video, music and other files
- **hachoir-urwid**: text user interface
- **hachoir-wx**: graphical user interface (wxWidgets)
- **hachoir-subfile**: find and extract all subfiles from any binary stream
- **hachoir-http**: HTML + Ajax user interface

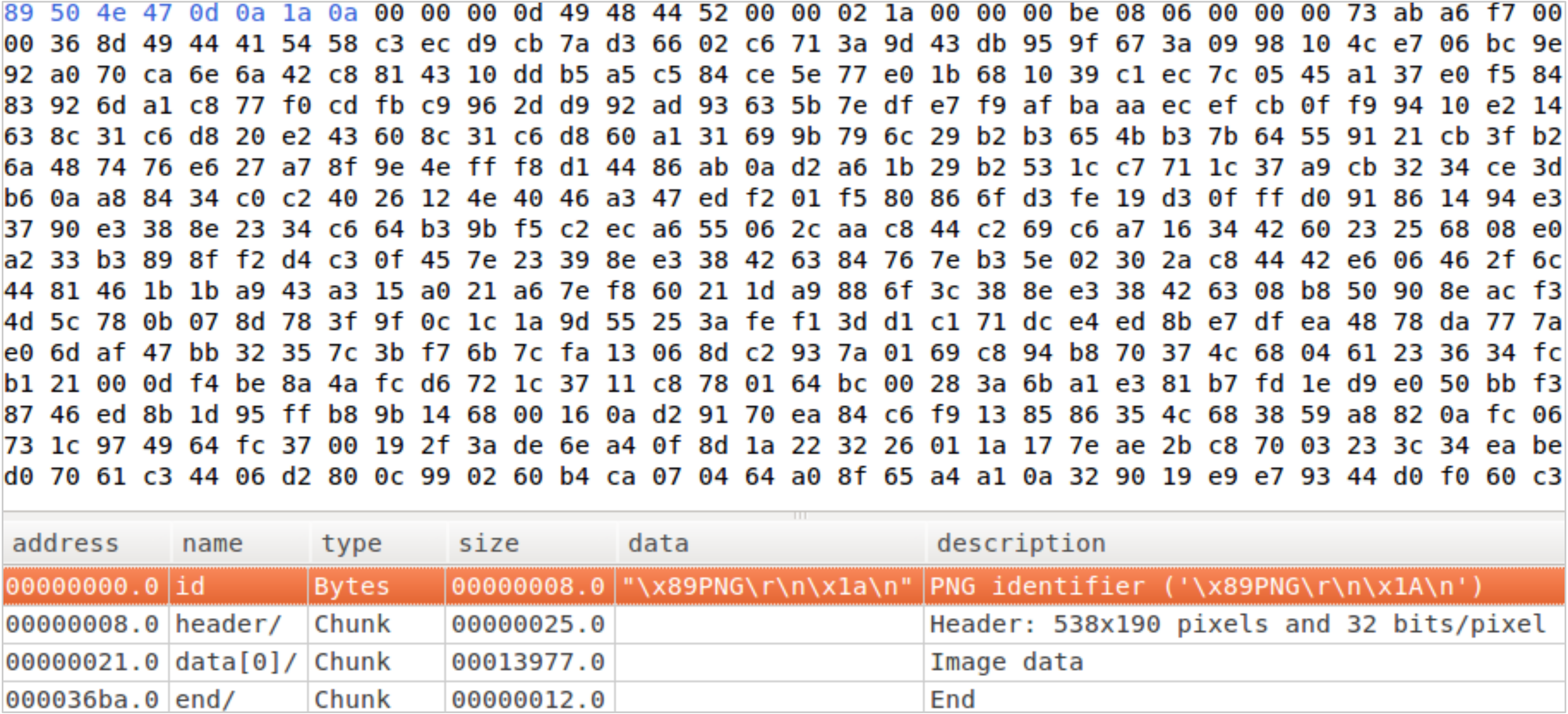
Modules:

- **hachoir-core**: see features and documentation
- **hachoir-parser**: list of supported file formats
- **hachoir-regex**: regular expression (regex) manipulation library

A tool for computer butchers.

(we'll use it from now on)

('hachoir' = meatgrinder)



```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 02 1a 00 00 00 be 08 06 00 00 00 73 ab a6 f7 00
00 36 8d 49 44 41 54 58 c3 ec d9 cb 7a d3 66 02 c6 71 3a 9d 43 db 95 9f 67 3a 09 98 10 4c e7 06 bc 9e
92 a0 70 ca 6e 6a 42 c8 81 43 10 dd b5 a5 c5 84 ce 5e 77 e0 1b 68 10 39 c1 ec 7c 05 45 a1 37 e0 f5 84
83 92 6d a1 c8 77 f0 cd fb c9 96 2d d9 92 ad 93 63 5b 7e df e7 f9 af ba aa ec ef cb 0f f9 94 10 e2 14
63 8c 31 c6 d8 20 e2 43 60 8c 31 c6 d8 60 a1 31 69 9b 79 6c 29 b2 b3 65 4b b3 7b 64 55 91 21 cb 3f b2
6a 48 74 76 e6 27 a7 8f 9e 4e ff f8 d1 44 86 ab 0a d2 a6 1b 29 b2 53 1c c7 71 1c 37 a9 cb 32 34 ce 3d
b6 0a a8 84 34 c0 c2 40 26 12 4e 40 46 a3 47 ed f2 01 f5 80 86 6f d3 fe 19 d3 0f ff d0 91 86 14 94 e3
37 90 e3 38 8e 23 34 c6 64 b3 9b f5 c2 ec a6 55 06 2c aa c8 44 c2 69 c6 a7 16 34 42 60 23 25 68 08 e0
a2 33 b3 89 8f f2 d4 c3 0f 45 7e 23 39 8e e3 38 42 63 84 76 7e b3 5e 02 30 2a c8 44 42 e6 06 46 2f 6c
44 81 46 1b 1b a9 43 a3 15 a0 21 a6 7e f8 60 21 1d a9 88 6f 3c 38 8e e3 38 42 63 08 b8 50 90 8e ac f3
4d 5c 78 0b 07 8d 78 3f 9f 0c 1c 1a 9d 55 25 3a fe f1 3d d1 c1 71 dc e4 ed 8b e7 df ea 48 78 da 77 7a
e0 6d af 47 bb 32 35 7c 3b f7 6b 7c fa 13 06 8d c2 93 7a 01 69 c8 94 b8 70 37 4c 68 04 61 23 36 34 fc
b1 21 00 0d f4 be 8a 4a fc d6 72 1c 37 11 c8 78 01 64 bc 00 28 3a 6b a1 e3 81 b7 fd 1e d9 e0 50 bb f3
87 46 ed 8b 1d 95 ff b8 9b 14 68 00 16 0a d2 91 70 ea 84 c6 f9 13 85 86 35 4c 68 38 59 a8 82 0a fc 06
73 1c 97 49 64 fc 37 00 19 2f 3a de 6e a4 0f 8d 1a 22 32 26 01 1a 17 7e ae 2b c8 70 03 23 3c 34 ea be
d0 70 61 c3 44 06 d2 80 0c 99 02 60 b4 ca 07 04 64 a0 8f 65 a4 a1 0a 32 90 19 e9 e7 93 44 d0 f0 60 c3
```

address	name	type	size	data	description
00000000.0	id	Bytes	00000008.0	"\x89PNG\r\n\x1a\n"	PNG identifier ('\x89PNG\r\n\x1A\n')
00000008.0	header/	Chunk	00000025.0		Header: 538x190 pixels and 32 bits/pixel
00000021.0	data[0]/	Chunk	00013977.0		Image data
000036ba.0	end/	Chunk	00000012.0		End

The Google logo, viewed in Hachoir:
a signature, then a sequence of chunks.

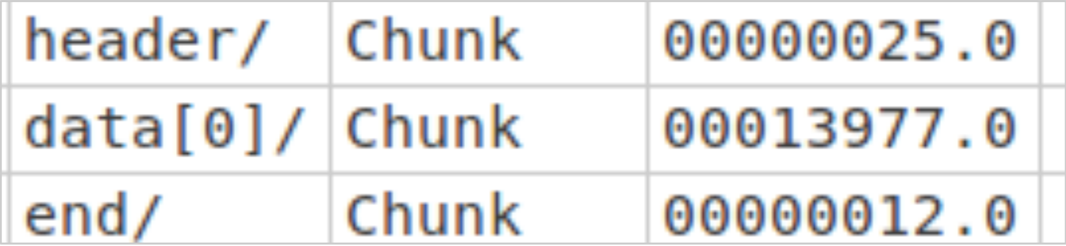
\x89 P N G \r \n ^Z \n

Compulsory signature at offset `0`

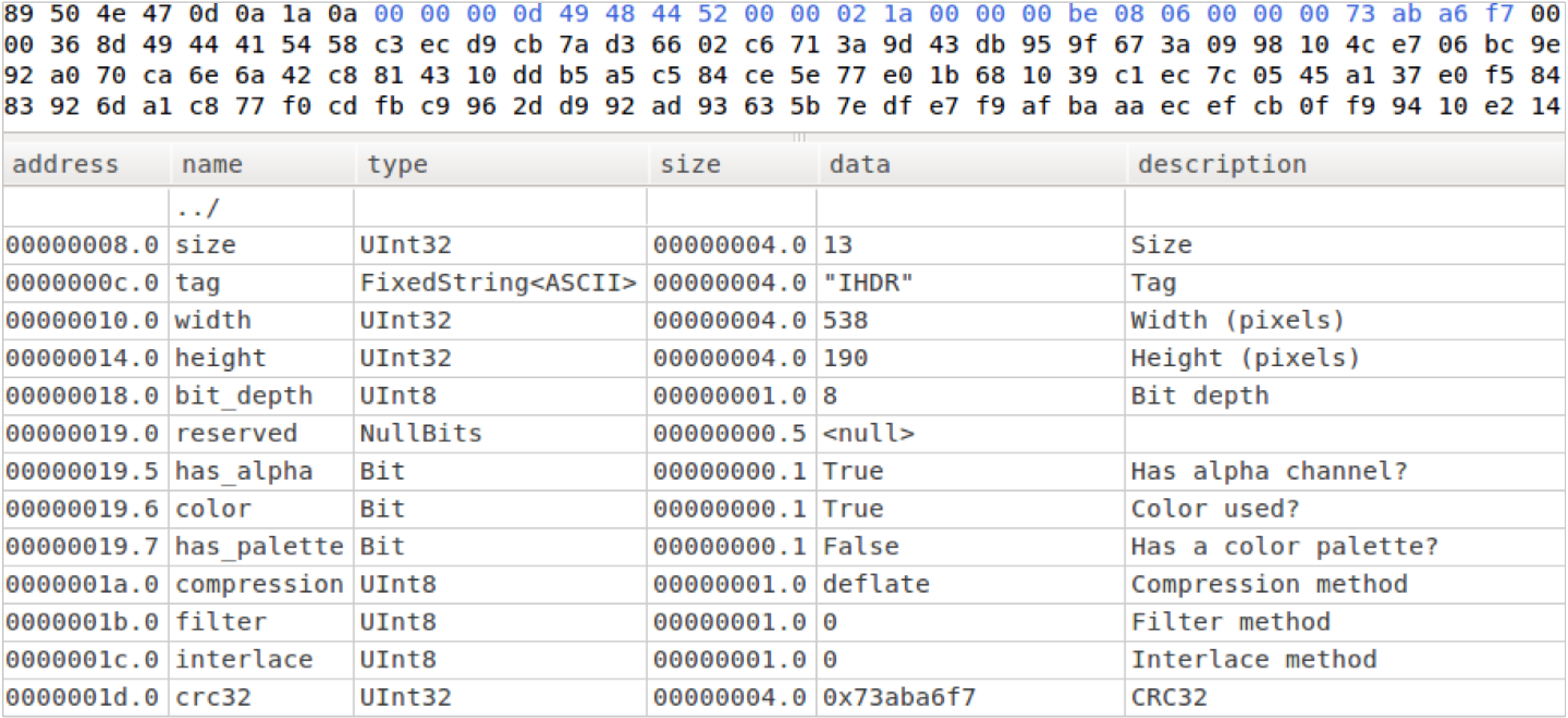
- identify the file type
- identify transfer errors
  - ○ \x89 : non ASCII (ASCII = [0 - 128])
  - ○ \r\n then \n : different end of line standards
  - ○ ^Z (\x1A) : “End Of File”

Chunk

- Common structure:
  a. size, on 4 bytes
  b. type, made of 4 letters
     - 1st letter: lowercase ⇒ ancillary chunk
  c. data
  d. checksum
     - CRC32(type + data)
- We can add custom chunks



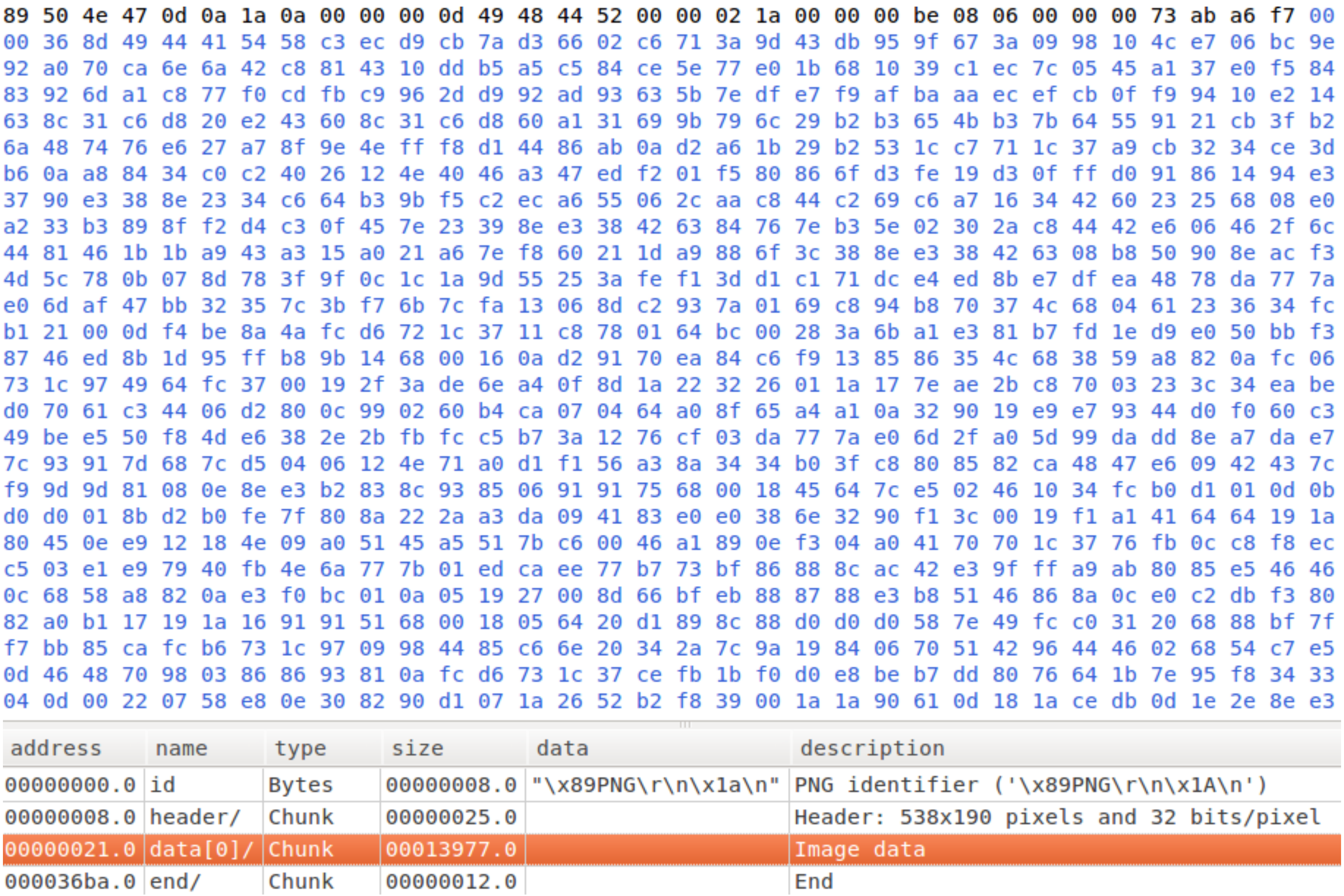
header/	Chunk	00000025.0
data[0]/	Chunk	00013977.0
end/	Chunk	00000012.0



```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 02 1a 00 00 00 be 08 06 00 00 00 73 ab a6 f7 00
00 36 8d 49 44 41 54 58 c3 ec d9 cb 7a d3 66 02 c6 71 3a 9d 43 db 95 9f 67 3a 09 98 10 4c e7 06 bc 9e
92 a0 70 ca 6e 6a 42 c8 81 43 10 dd b5 a5 c5 84 ce 5e 77 e0 1b 68 10 39 c1 ec 7c 05 45 a1 37 e0 f5 84
83 92 6d a1 c8 77 f0 cd fb c9 96 2d d9 92 ad 93 63 5b 7e df e7 f9 af ba aa ec ef cb 0f f9 94 10 e2 14
```

address	name	type	size	data	description
	../				
00000008.0	size	UInt32	00000004.0	13	Size
0000000c.0	tag	FixedString<ASCII>	00000004.0	"IHDR"	Tag
00000010.0	width	UInt32	00000004.0	538	Width (pixels)
00000014.0	height	UInt32	00000004.0	190	Height (pixels)
00000018.0	bit_depth	UInt8	00000001.0	8	Bit depth
00000019.0	reserved	NullBits	00000000.5	<null>	
00000019.5	has_alpha	Bit	00000000.1	True	Has alpha channel?
00000019.6	color	Bit	00000000.1	True	Color used?
00000019.7	has_palette	Bit	00000000.1	False	Has a color palette?
0000001a.0	compression	UInt8	00000001.0	deflate	Compression method
0000001b.0	filter	UInt8	00000001.0	0	Filter method
0000001c.0	interlace	UInt8	00000001.0	0	Interlace method
0000001d.0	crc32	UInt32	00000004.0	0x73aba6f7	CRC32

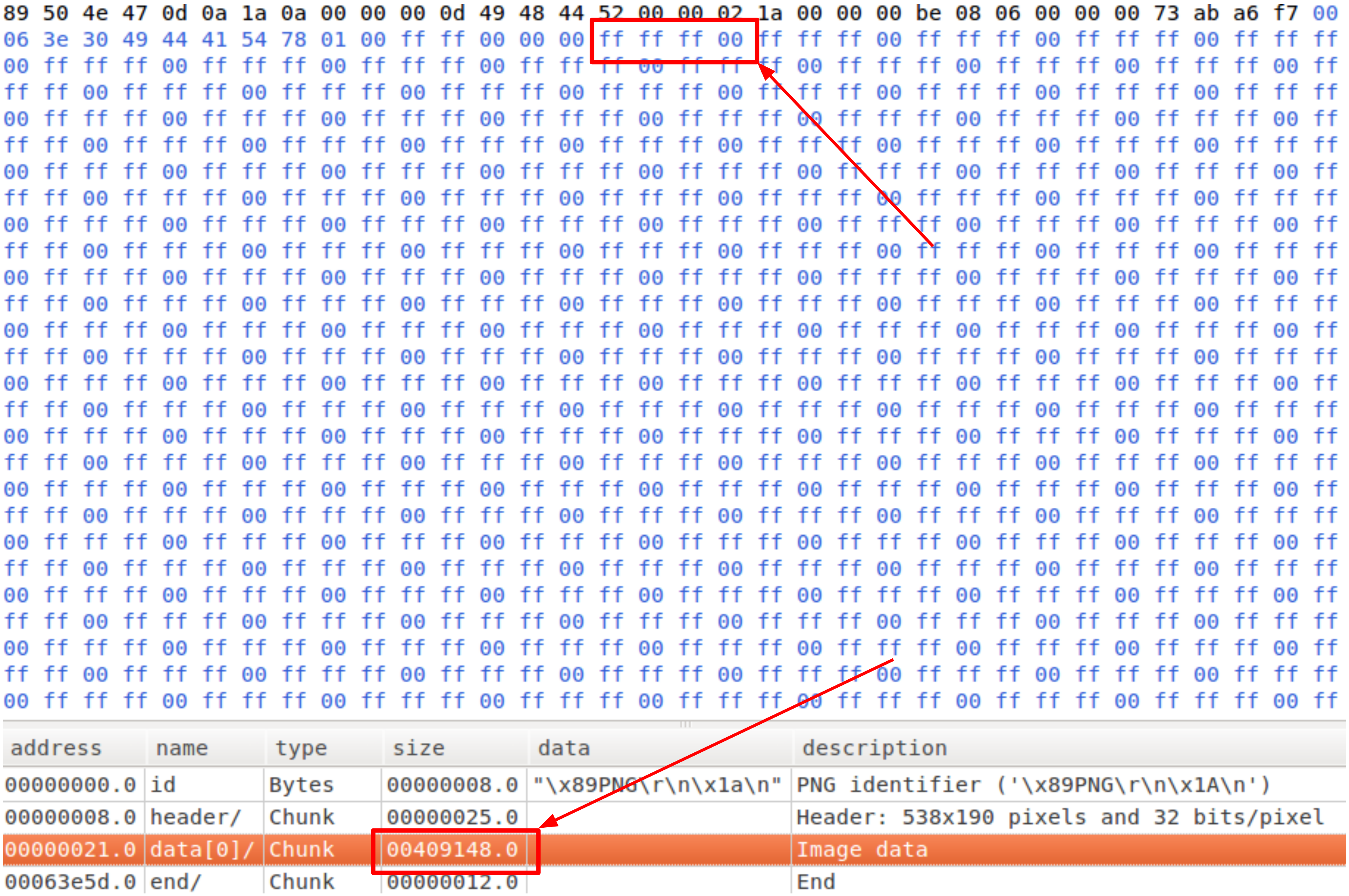
IHDR chunk: containing image information



```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 02 1a 00 00 00 be 08 06 00 00 00 73 ab a6 f7 00
00 36 8d 49 44 41 54 58 c3 ec d9 cb 7a d3 66 02 c6 71 3a 9d 43 db 95 9f 67 3a 09 98 10 4c e7 06 bc 9e
92 a0 70 ca 6e 6a 42 c8 81 43 10 dd b5 a5 c5 84 ce 5e 77 e0 1b 68 10 39 c1 ec 7c 05 45 a1 37 e0 f5 84
83 92 6d a1 c8 77 f0 cd fb c9 96 2d d9 92 ad 93 63 5b 7e df e7 f9 af ba aa ec ef cb 0f f9 94 10 e2 14
63 8c 31 c6 d8 20 e2 43 60 8c 31 c6 d8 60 a1 31 69 9b 79 6c 29 b2 b3 65 4b b3 7b 64 55 91 21 cb 3f b2
6a 48 74 76 e6 27 a7 8f 9e 4e ff f8 d1 44 86 ab 0a d2 a6 1b 29 b2 53 1c c7 71 1c 37 a9 cb 32 34 ce 3d
b6 0a a8 84 34 c0 c2 40 26 12 4e 40 46 a3 47 ed f2 01 f5 80 86 6f d3 fe 19 d3 0f ff d0 91 86 14 94 e3
37 90 e3 38 8e 23 34 c6 64 b3 9b f5 c2 ec a6 55 06 2c aa c8 44 c2 69 c6 a7 16 34 42 60 23 25 68 08 e0
a2 33 b3 89 8f f2 d4 c3 0f 45 7e 23 39 8e e3 38 42 63 84 76 7e b3 5e 02 30 2a c8 44 42 e6 06 46 2f 6c
44 81 46 1b 1b a9 43 a3 15 a0 21 a6 7e f8 60 21 1d a9 88 6f 3c 38 8e e3 38 42 63 08 b8 50 90 8e ac f3
4d 5c 78 0b 07 8d 78 3f 9f 0c 1c 1a 9d 55 25 3a fe f1 3d d1 c1 71 dc e4 ed 8b e7 df ea 48 78 da 77 7a
e0 6d af 47 bb 32 35 7c 3b f7 6b 7c fa 13 06 8d c2 93 7a 01 69 c8 94 b8 70 37 4c 68 04 61 23 36 34 fc
b1 21 00 0d f4 be 8a 4a fc d6 72 1c 37 11 c8 78 01 64 bc 00 28 3a 6b a1 e3 81 b7 fd 1e d9 e0 50 bb f3
87 46 ed 8b 1d 95 ff b8 9b 14 68 00 16 0a d2 91 70 ea 84 c6 f9 13 85 86 35 4c 68 38 59 a8 82 0a fc 06
73 1c 97 49 64 fc 37 00 19 2f 3a de 6e a4 0f 8d 1a 22 32 26 01 1a 17 7e ae 2b c8 70 03 23 3c 34 ea be
d0 70 61 c3 44 06 d2 80 0c 99 02 60 b4 ca 07 04 64 a0 8f 65 a4 a1 0a 32 90 19 e9 e7 93 44 d0 f0 60 c3
49 be e5 50 f8 4d e6 38 2e 2b fb fc c5 b7 3a 12 76 cf 03 da 77 7a e0 6d 2f a0 5d 99 da dd 8e a7 da e7
7c 93 91 7d 68 7c d5 04 06 12 4e 71 a0 d1 f1 56 a3 8a 34 34 b0 3f c8 80 85 82 ca 48 47 e6 09 42 43 7c
f9 9d 9d 81 08 0e 8e e3 b2 83 8c 93 85 06 91 91 75 68 00 18 45 64 7c e5 02 46 10 34 fc b0 d1 01 0d 0b
d0 d0 01 8b d2 b0 fe 7f 80 8a 22 2a a3 da 09 41 83 e0 e0 38 6e 32 90 f1 3c 00 19 f1 a1 41 64 64 19 1a
80 45 0e e9 12 18 4e 09 a0 51 45 a5 51 7b c6 00 46 a1 89 0e f3 04 a0 41 70 70 1c 37 76 fb 0c c8 f8 ec
c5 03 e1 e9 79 40 fb 4e 6a 77 7b 01 ed ca ee 77 b7 73 bf 86 88 8c ac 42 e3 9f ff a9 ab 80 85 e5 46 46
0c 68 58 a8 82 0a e3 f0 bc 01 0a 05 19 27 00 8d 66 bf eb 88 87 88 e3 b8 51 46 86 8a 0c e0 c2 db f3 80
82 a0 b1 17 19 1a 16 91 91 51 68 00 18 05 64 20 d1 89 8c 88 d0 d0 d0 58 7e 49 fc c0 31 20 68 88 bf 7f
f7 bb 85 ca fc b6 73 1c 97 09 98 44 85 c6 6e 20 34 2a 7c 9a 19 84 06 70 51 42 96 44 46 02 68 54 c7 e5
0d 46 48 70 98 03 86 86 93 81 0a fc d6 73 1c 37 ce fb 1b f0 d0 e8 be b7 dd 80 76 64 1b 7e 95 f8 34 33
04 0d 00 22 07 58 e8 0e 30 82 90 d1 07 1a 26 52 b2 f8 39 00 1a 1a 90 61 0d 18 1a ce db 0d 1e 2e 8e e3
```

address	name	type	size	data	description
00000000.0	id	Bytes	00000008.0	"\x89PNG\r\n\x1a\n"	PNG identifier ('\x89PNG\r\n\x1A\n')
00000008.0	header/	Chunk	00000025.0		Header: 538x190 pixels and 32 bits/pixel
00000021.0	data[0]/	Chunk	00013977.0		Image data
000036ba.0	end/	Chunk	00000012.0		End

IDAT chunk (compressed): pixels values

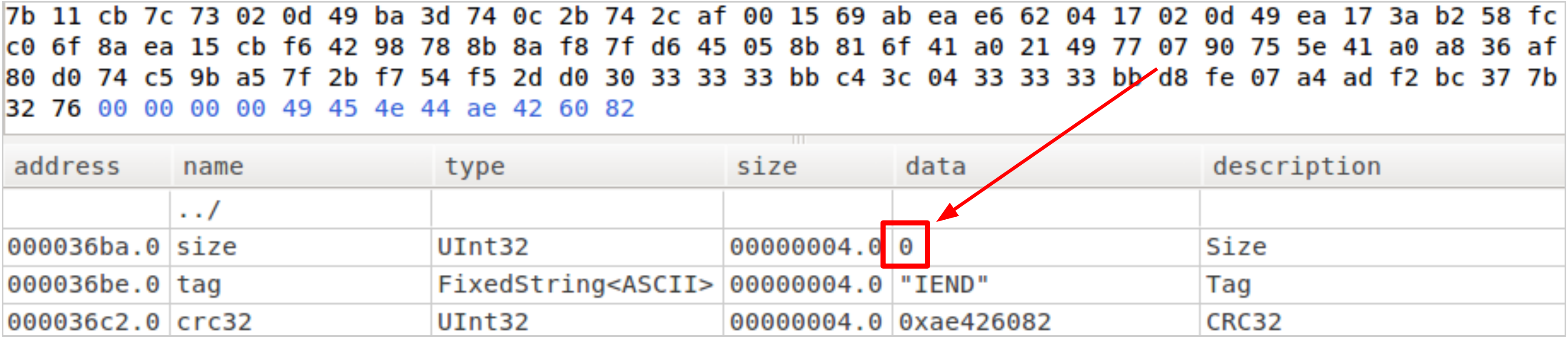


```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 02 1a 00 00 00 be 08 06 00 00 00 73 ab a6 f7 00
06 3e 30 49 44 41 54 78 01 00 ff ff 00 00 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff
00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff ff ff 00 ff
```

address	name	type	size	data	description
00000000.0	id	Bytes	00000008.0	"\x89PNG\r\n\x1a\n"	PNG identifier ('\x89PNG\r\n\x1A\n')
00000008.0	header/	Chunk	00000025.0		Header: 538x190 pixels and 32 bits/pixel
00000021.0	data[0]/	Chunk	00409148.0		Image data
00063e5d.0	end/	Chunk	00000012.0		End

IDAT after decompression

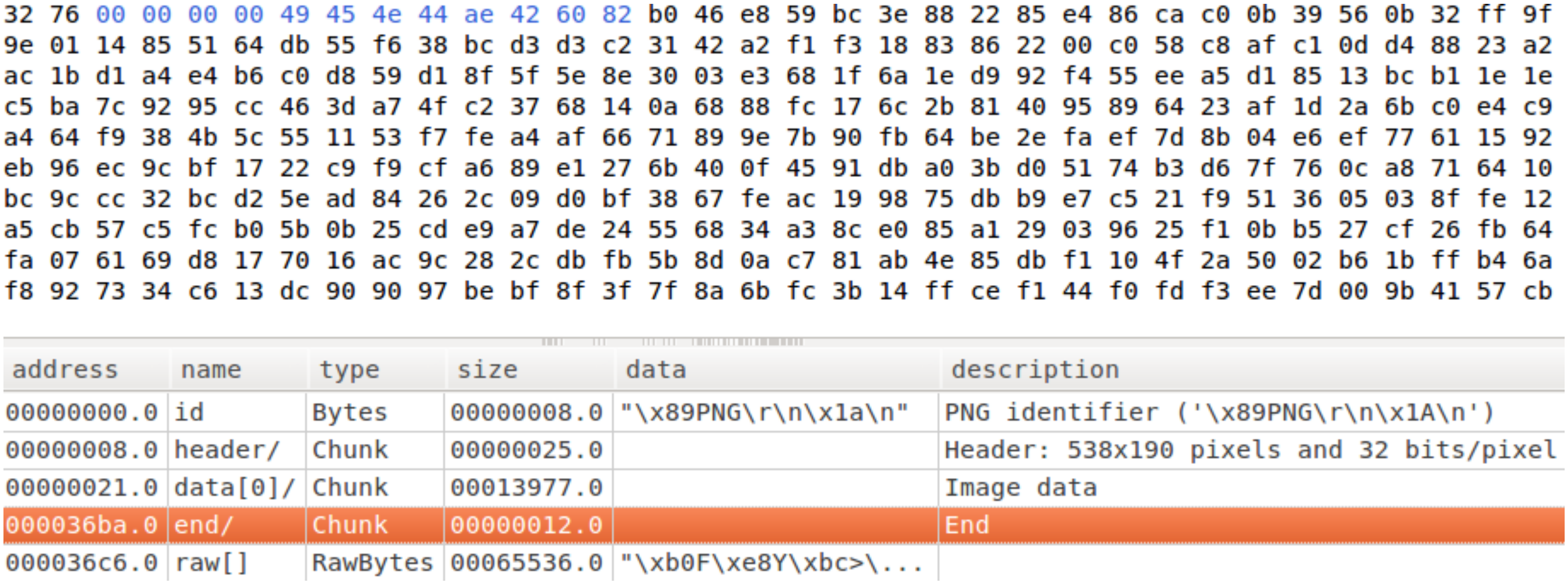
(FF FF FF 00 = black + 100% transparent)



```
7b 11 cb 7c 73 02 0d 49 ba 3d 74 0c 2b 74 2c af 00 15 69 ab ea e6 62 04 17 02 0d 49 ea 17 3a b2 58 fc
c0 6f 8a ea 15 cb f6 42 98 78 8b 8a f8 7f d6 45 05 8b 81 6f 41 a0 21 49 77 07 90 75 5e 41 a0 a8 36 af
80 d0 74 c5 9b a5 7f 2b f7 54 f5 2d d0 30 33 33 33 bb c4 3c 04 33 33 33 bb d8 fe 07 a4 ad f2 bc 37 7b
32 76 00 00 00 00 49 45 4e 44 ae 42 60 82
```

address	name	type	size	data	description
	../				
000036ba.0	size	UInt32	00000004.0	0	Size
000036be.0	tag	FixedString<ASCII>	00000004.0	"IEND"	Tag
000036c2.0	crc32	UInt32	00000004.0	0xae426082	CRC32

IEND chunk: End of File (‘s structure)



```
32 76 00 00 00 00 49 45 4e 44 ae 42 60 82 b0 46 e8 59 bc 3e 88 22 85 e4 86 ca c0 0b 39 56 0b 32 ff 9f
9e 01 14 85 51 64 db 55 f6 38 bc d3 d3 c2 31 42 a2 f1 f3 18 83 86 22 00 c0 58 c8 af c1 0d d4 88 23 a2
ac 1b d1 a4 e4 b6 c0 d8 59 d1 8f 5f 5e 8e 30 03 e3 68 1f 6a 1e d9 92 f4 55 ee a5 d1 85 13 bc b1 1e 1e
c5 ba 7c 92 95 cc 46 3d a7 4f c2 37 68 14 0a 68 88 fc 17 6c 2b 81 40 95 89 64 23 af 1d 2a 6b c0 e4 c9
a4 64 f9 38 4b 5c 55 11 53 f7 fe a4 af 66 71 89 9e 7b 90 fb 64 be 2e fa ef 7d 8b 04 e6 ef 77 61 15 92
eb 96 ec 9c bf 17 22 c9 f9 cf a6 89 e1 27 6b 40 0f 45 91 db a0 3b d0 51 74 b3 d6 7f 76 0c a8 71 64 10
bc 9c cc 32 bc d2 5e ad 84 26 2c 09 d0 bf 38 67 fe ac 19 98 75 db b9 e7 c5 21 f9 51 36 05 03 8f fe 12
a5 cb 57 c5 fc b0 5b 0b 25 cd e9 a7 de 24 55 68 34 a3 8c e0 85 a1 29 03 96 25 f1 0b b5 27 cf 26 fb 64
fa 07 61 69 d8 17 70 16 ac 9c 28 2c db fb 5b 8d 0a c7 81 ab 4e 85 db f1 10 4f 2a 50 02 b6 1b ff b4 6a
f8 92 73 34 c6 13 dc 90 90 97 be bf 8f 3f 7f 8a 6b fc 3b 14 ff ce f1 44 f0 fd f3 ee 7d 00 9b 41 57 cb
```

address	name	type	size	data	description
00000000.0	id	Bytes	00000008.0	"\x89PNG\r\n\x1a\n"	PNG identifier ('\x89PNG\r\n\x1A\n')
00000008.0	header/	Chunk	00000025.0		Header: 538x190 pixels and 32 bits/pixel
00000021.0	data[0]/	Chunk	00013977.0		Image data
000036ba.0	end/	Chunk	00000012.0		End
000036c6.0	raw[]	RawBytes	00065536.0	"\xb0F\xe8Y\xbc>\...	

What comes after IEND is ignored by PNG tools.
(the image is complete)

Recap

Structure:

1. Signature at offset `0`
2. Chunks sequence
   a. `IHDR` header
   b. `IDAT` data
   c. `IEND` end

“I know how Google works!”

Now, you can impress your friends!

ëPNG♪◙→◙   ♪IHDR ✓

(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)

+

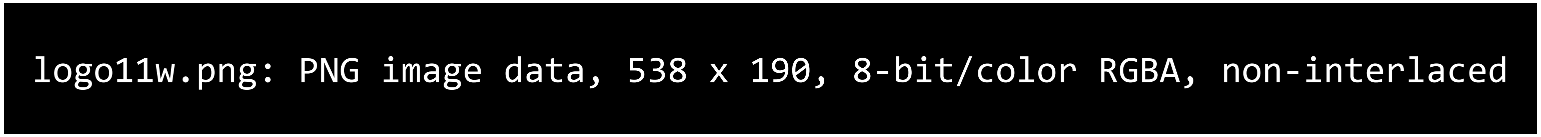
MySecretKey12345

↓

:¼N?â?pzILá+?ìgU ✗

(3A AC 4E 10 83 03 70 7A 49 4C A0 DA 0B 8D 67 55)

Encryption breaks the signature.

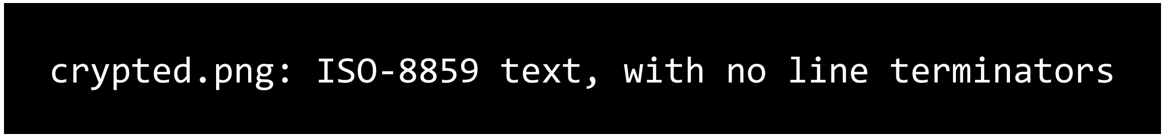


```
logo11w.png: PNG image data, 538 x 190, 8-bit/color RGBA, non-interlaced
```

✓

+

MySecretKey12345



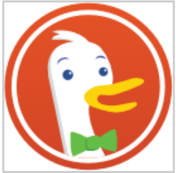
```
crypted.png: ISO-8859 text, with no line terminators
```

✗

Without a signature, the encrypted file is invalid.

If we encrypt a PNG,
we don’t get a PNG

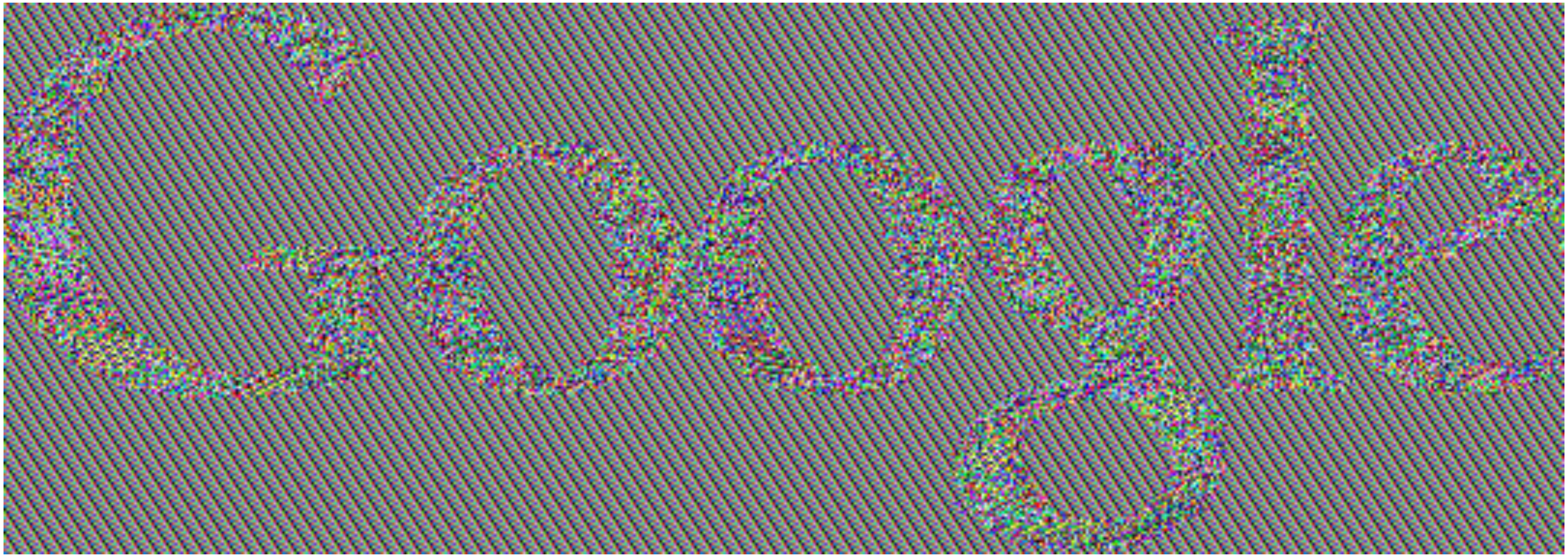
the signature is broken and the structure too
(***a priori***)

How can we encrypt Google into  ?

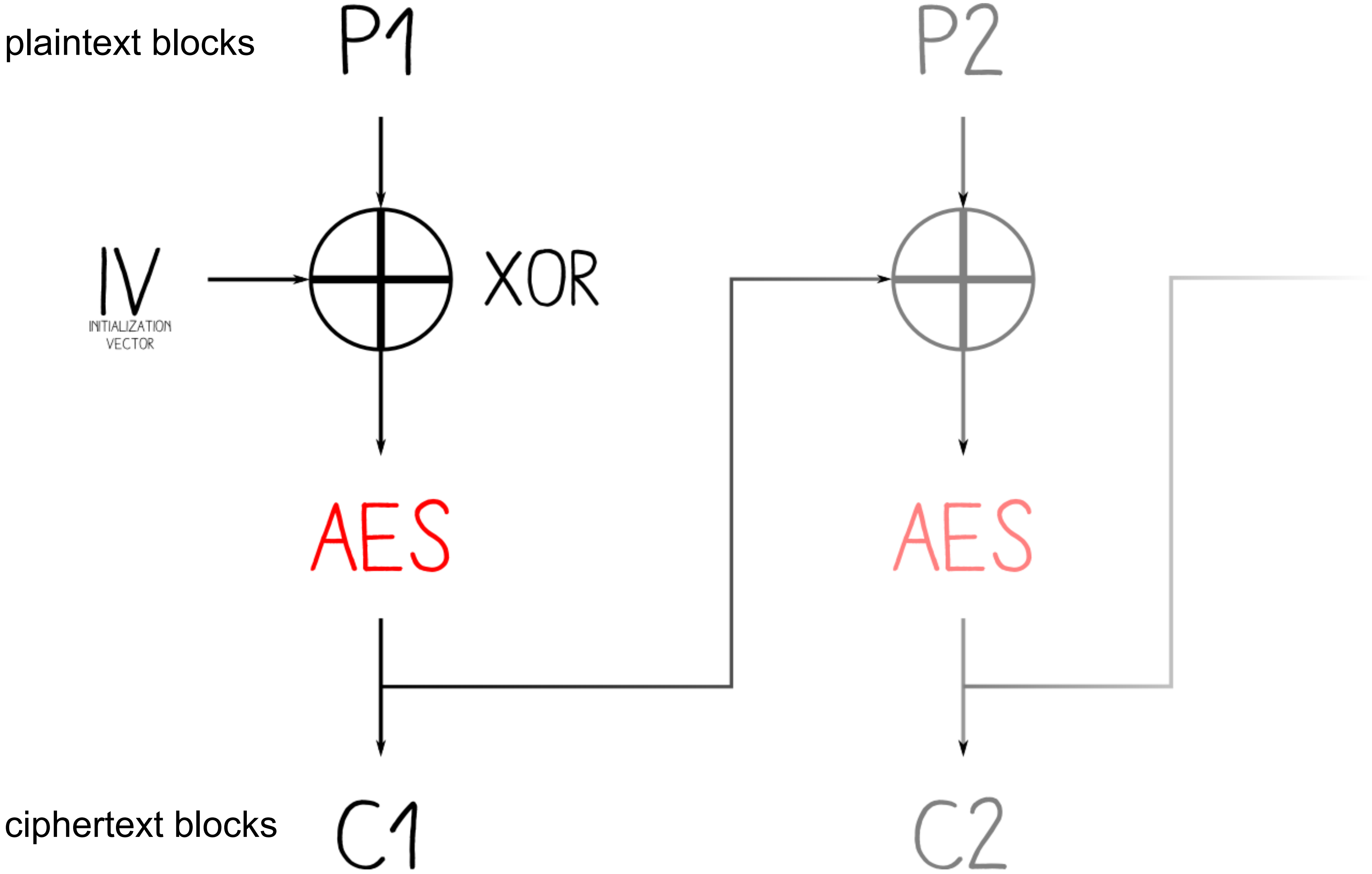
How can we control input *and* output?

AES works with blocks

How can we use it on a file?



What happens if each block of a file is encrypted independently (ECB mode)



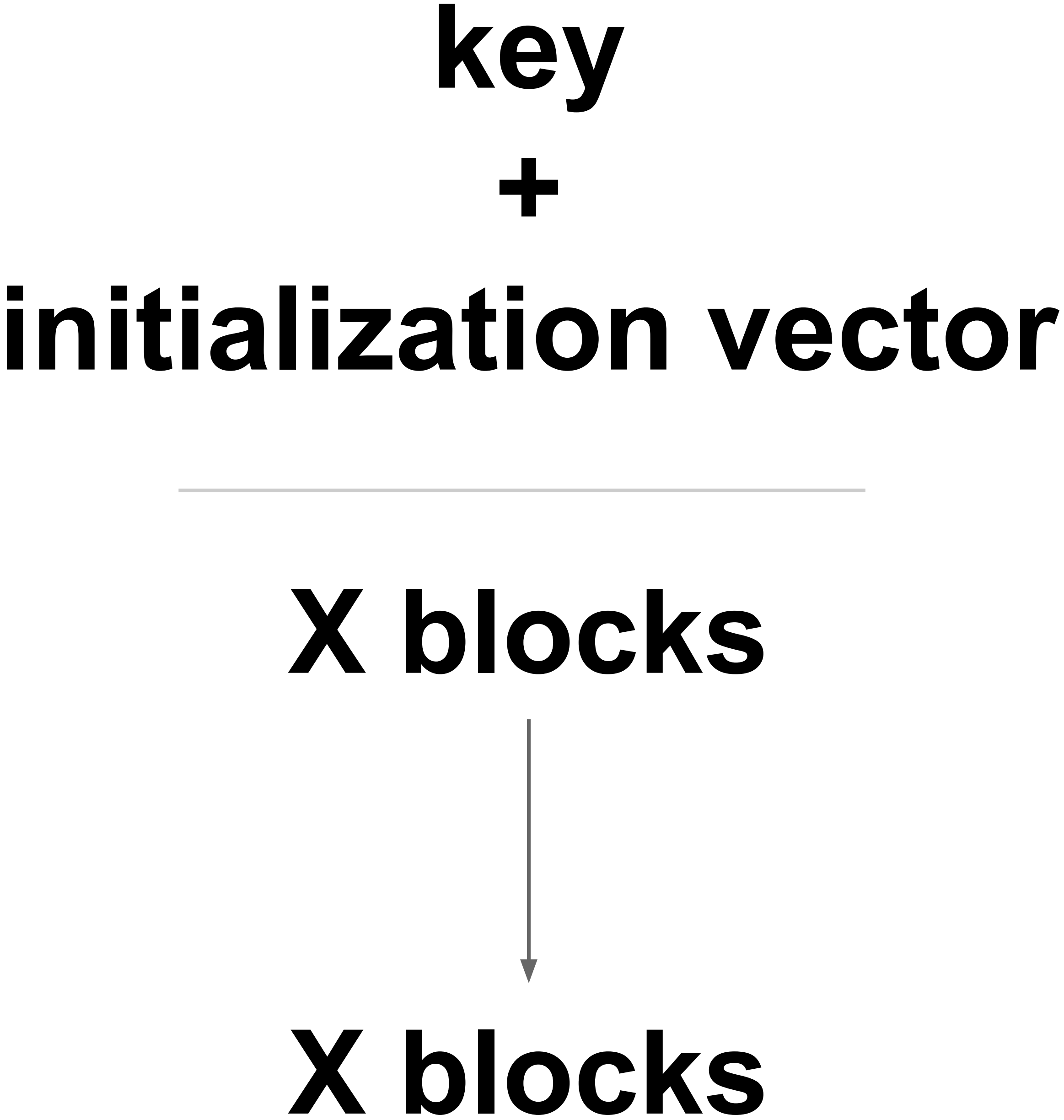
plaintext blocks

ciphertext blocks

In CBC mode, each encryption depends on previous blocks.

The CBC mode

- “Cipher Block Chaining”
  - considered secure
  - we’ll use it from now on
- introduces an Initialization Vector
  - extra parameter
  - arbitrary
  - in practice, it should be unpredictable



key

+

initialization vector

X blocks

X blocks

Relations between blocks and IV

$C1 = \text{ENC}^{*}(IV \wedge P1)$

* we use the same key for all operations.

Relations between blocks and IV

C1 = ENC(IV ^ P1)

DEC(C1) = DEC(ENC(IV ^ P1))

Decrypt both sides...

Relations between blocks and IV

$$\text{C1} = \text{ENC}(\text{IV} \wedge \text{P1})$$

$$\text{DEC}(\text{C1}) = \cancel{\text{DEC}}(\cancel{\text{ENC}}(\text{IV} \wedge \text{P1}))$$

it cancels itself

Relations between blocks and IV

C1 = ENC(IV ^ P1)

DEC(C1) = IV ^ P1

Relations between blocks and IV

$$\begin{aligned} C1 &= \mathrm{ENC}(IV \wedge P1) \\ \mathrm{DEC}(C1) &= IV \wedge P1 \\ P1 \wedge \mathrm{DEC}(C1) &= IV \wedge P1 \wedge P1 \end{aligned}$$

Apply a XOR on both sides...

Relations between blocks and IV

C1 = ENC(IV ^ P1)

DEC(C1) = IV ^ P1

P1 ^ DEC(C1) = IV ~~^ P1 ^ P1~~

...it cancels itself

Relations between blocks and IV

$$C1 = ENC(IV \wedge P1)$$

$$DEC(C1) = IV \wedge P1$$

$$P1 \wedge DEC(C1) = IV$$

Relations between blocks and IV

$$C1 = ENC(IV \wedge P1)$$

$$DEC(C1) = IV \wedge P1$$

$$P1 \wedge DEC(C1) = IV$$

$$\Rightarrow IV = P1 \wedge DEC(C1)$$

We get a relation of IV from P1 and C1

IV = P1 ^ DEC(C1)

P1, C1 are the first 16 bytes of our 2 files

once the key ***k*** is chosen,

1. decrypt C1
2. apply a XOR with P1

⇒ we get the IV

k **key**

+

IV **initialization vector**

Px **X blocks**

↓

Cx **X blocks**

k

IV

P1 **ëPNG♪◙→◙   ♪IHDR**

(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)

C1 **ëPNG♪◙→◙   ♪IHDR**

(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)

k `IVManipulation!!`

IV

P1 **ëPNG♪◙→◙   ♪IHDR**

`(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)`

C1 **ëPNG♪◙→◙   ♪IHDR**

`(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)`

k	`IVManipulation!!`
IV	`P1 ^ DEC(C1)`
P1	**ëPNG♪◙→◙ ♪IHDR** `(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)`
C1	**ëPNG♪◙→◙ ♪IHDR** `(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)`

k IVManipulation!!

IV r 1ÿ4┼╬ ╕ ·§{ú)u≡

(72 00 31 98 34 C5 CE 00 B8 FA 15 7B A3 29 75 F0)

P1 ëPNG♪◙→◙   ♪IHDR

(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)

C1 ëPNG♪◙→◙   ♪IHDR

(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)

k IVManipulation!!

+

IV r 1ÿ4┼╬ ╕·§{ú)u≡

(72 00 31 98 34 C5 CE 00 B8 FA 15 7B A3 29 75 F0)

P1 ëPNG♪◙→◙   ♪IHDR ✓

(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)

↓

C1 ëPNG♪◙→◙   ♪IHDR ✓

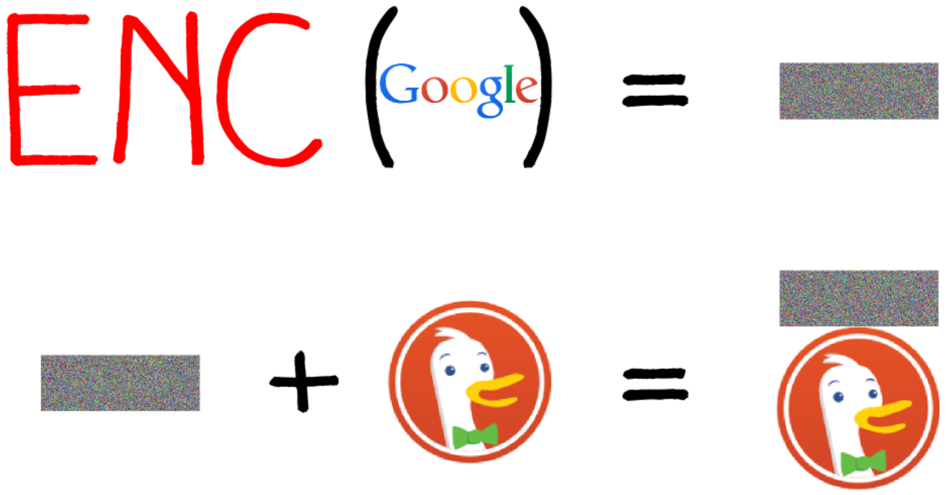
(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)

Status

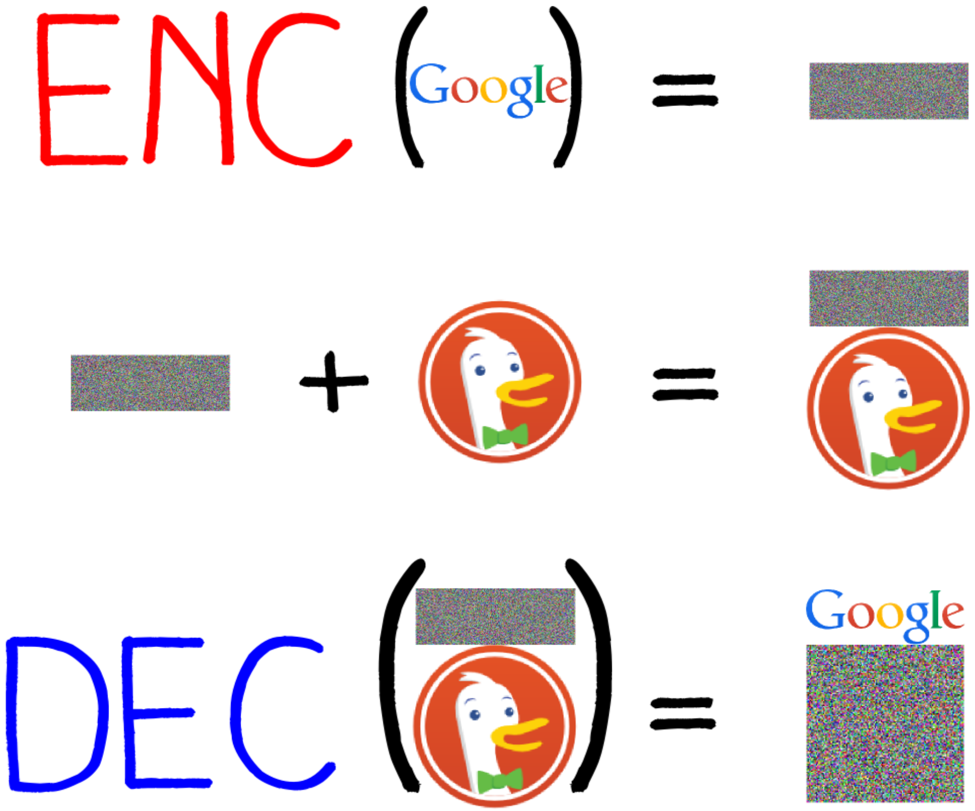
- we control the first cipher block
    - we can get a valid signature
        - and 8 extra bytes
- we control nothing else
    - no valid structure

How can we control
the structure via encryption?

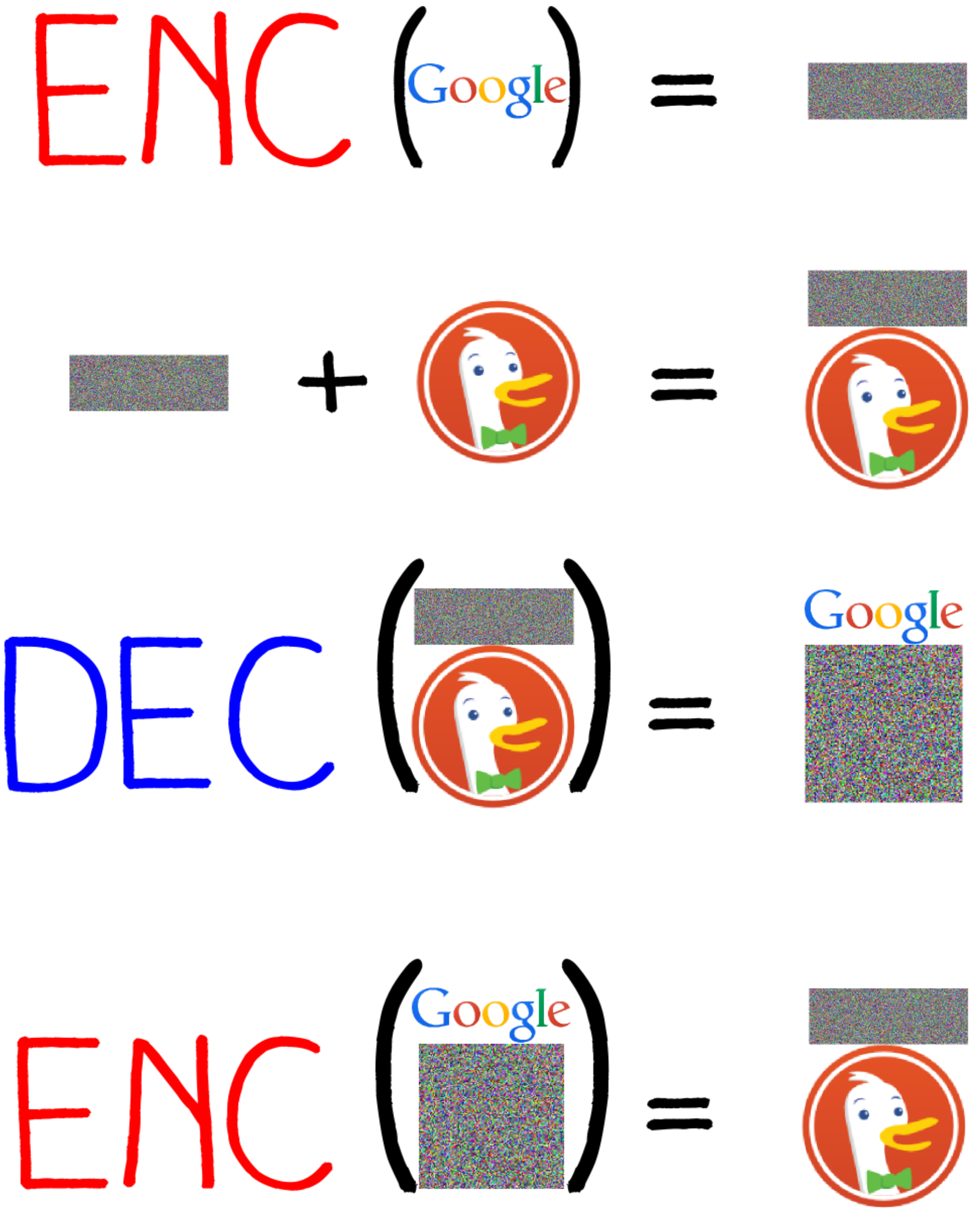
If we encrypt our picture, we get random data.



If we append another picture to this random data...



… we get back our original picture after decryption.
(followed by some different random data)



If we encrypt the final result, we get our first random data, followed by our target picture.

Pre-decrypt data

- Decrypt our target’s chunks
- Append them to our source file
  - at the start of the next block
    (*pad* if necessary)
  - it’s still valid thanks to the `IEND` chunk

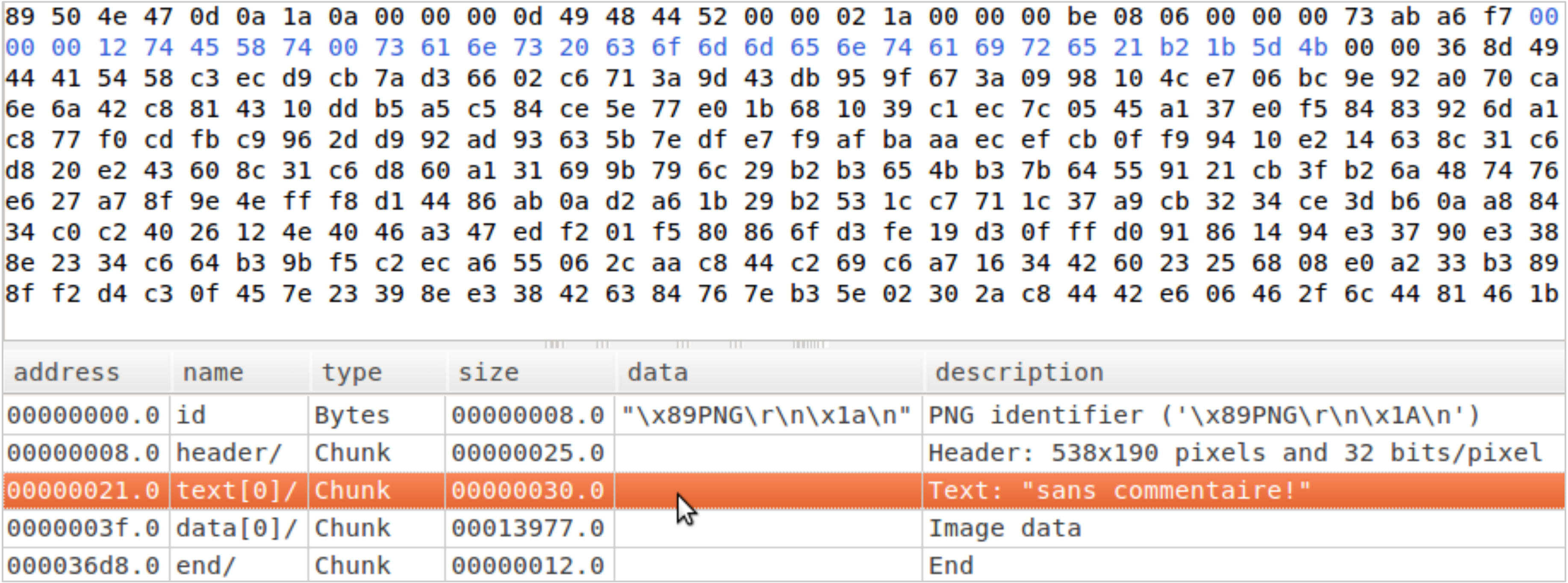
Status

- We control
  - a bit of the input
  - a bit of the output
- The source file is still valid
  - original source file (valid)
  - followed by decrypted data

How can we control crypted data ?

We won't ☺

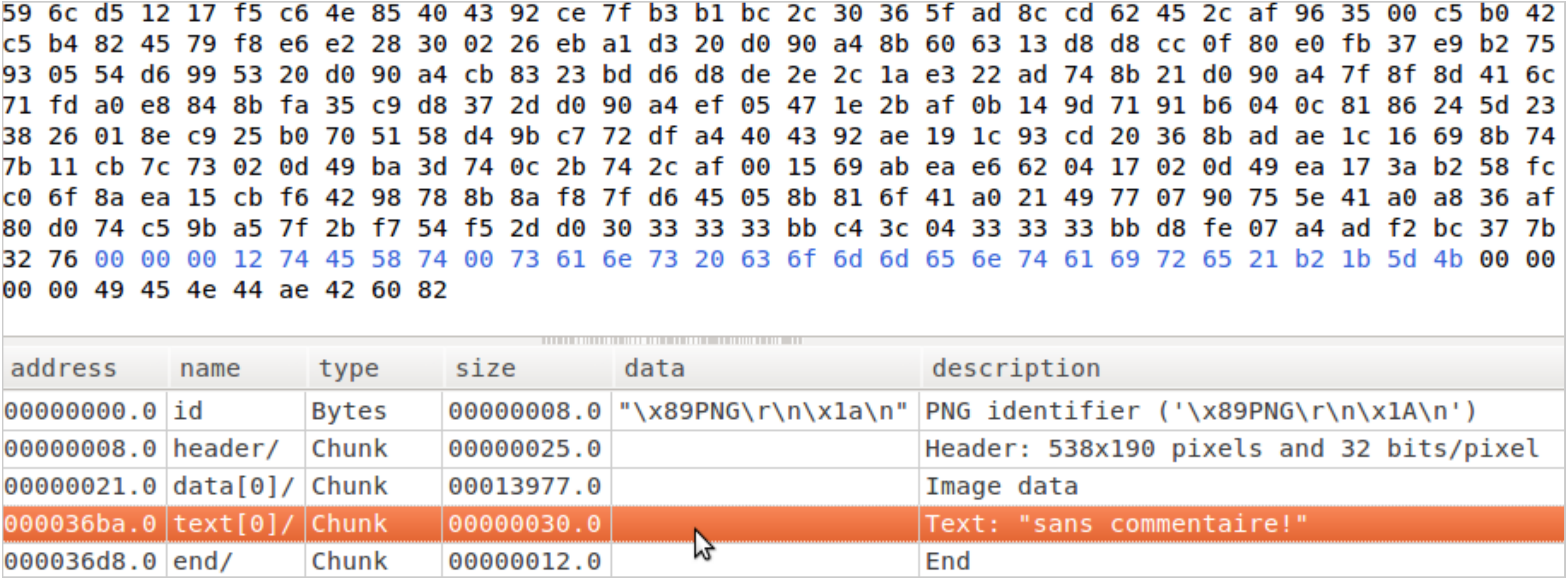
We'll ask the file format to ignore it.



```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 02 1a 00 00 00 be 08 06 00 00 00 73 ab a6 f7 00
00 00 12 74 45 58 74 00 73 61 6e 73 20 63 6f 6d 6d 65 6e 74 61 69 72 65 21 b2 1b 5d 4b 00 00 36 8d 49
44 41 54 58 c3 ec d9 cb 7a d3 66 02 c6 71 3a 9d 43 db 95 9f 67 3a 09 98 10 4c e7 06 bc 9e 92 a0 70 ca
6e 6a 42 c8 81 43 10 dd b5 a5 c5 84 ce 5e 77 e0 1b 68 10 39 c1 ec 7c 05 45 a1 37 e0 f5 84 83 92 6d a1
c8 77 f0 cd fb c9 96 2d d9 92 ad 93 63 5b 7e df e7 f9 af ba aa ec ef cb 0f f9 94 10 e2 14 63 8c 31 c6
d8 20 e2 43 60 8c 31 c6 d8 60 a1 31 69 9b 79 6c 29 b2 b3 65 4b b3 7b 64 55 91 21 cb 3f b2 6a 48 74 76
e6 27 a7 8f 9e 4e ff f8 d1 44 86 ab 0a d2 a6 1b 29 b2 53 1c c7 71 1c 37 a9 cb 32 34 ce 3d b6 0a a8 84
34 c0 c2 40 26 12 4e 40 46 a3 47 ed f2 01 f5 80 86 6f d3 fe 19 d3 0f ff d0 91 86 14 94 e3 37 90 e3 38
8e 23 34 c6 64 b3 9b f5 c2 ec a6 55 06 2c aa c8 44 c2 69 c6 a7 16 34 42 60 23 25 68 08 e0 a2 33 b3 89
8f f2 d4 c3 0f 45 7e 23 39 8e e3 38 42 63 84 76 7e b3 5e 02 30 2a c8 44 42 e6 06 46 2f 6c 44 81 46 1b
```

address	name	type	size	data	description
00000000.0	id	Bytes	00000008.0	"\x89PNG\r\n\x1a\n"	PNG identifier ('\x89PNG\r\n\x1A\n')
00000008.0	header/	Chunk	00000025.0		Header: 538x190 pixels and 32 bits/pixel
00000021.0	text[0]/	Chunk	00000030.0		Text: "sans commentaire!"
0000003f.0	data[0]/	Chunk	00013977.0		Image data
000036d8.0	end/	Chunk	00000012.0		End

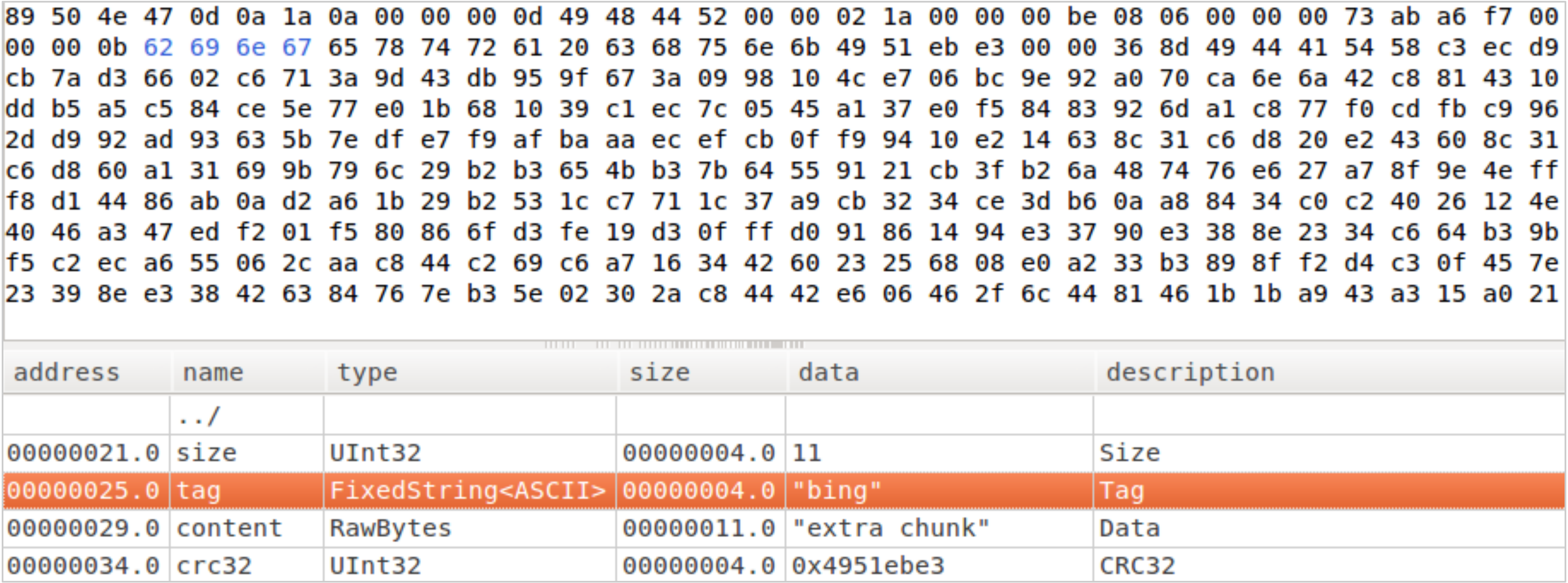
Adding a standard comment chunk (tEXt type)



```
59 6c d5 12 17 f5 c6 4e 85 40 43 92 ce 7f b3 b1 bc 2c 30 36 5f ad 8c cd 62 45 2c af 96 35 00 c5 b0 42
c5 b4 82 45 79 f8 e6 e2 28 30 02 26 eb a1 d3 20 d0 90 a4 8b 60 63 13 d8 d8 cc 0f 80 e0 fb 37 e9 b2 75
93 05 54 d6 99 53 20 d0 90 a4 cb 83 23 bd d6 d8 de 2e 2c 1a e3 22 ad 74 8b 21 d0 90 a4 7f 8f 8d 41 6c
71 fd a0 e8 84 8b fa 35 c9 d8 37 2d d0 90 a4 ef 05 47 1e 2b af 0b 14 9d 71 91 b6 04 0c 81 86 24 5d 23
38 26 01 8e c9 25 b0 70 51 58 d4 9b c7 72 df a4 40 43 92 ae 19 1c 93 cd 20 36 8b ad ae 1c 16 69 8b 74
7b 11 cb 7c 73 02 0d 49 ba 3d 74 0c 2b 74 2c af 00 15 69 ab ea e6 62 04 17 02 0d 49 ea 17 3a b2 58 fc
c0 6f 8a ea 15 cb f6 42 98 78 8b 8a f8 7f d6 45 05 8b 81 6f 41 a0 21 49 77 07 90 75 5e 41 a0 a8 36 af
80 d0 74 c5 9b a5 7f 2b f7 54 f5 2d d0 30 33 33 33 bb c4 3c 04 33 33 33 bb d8 fe 07 a4 ad f2 bc 37 7b
32 76 00 00 00 12 74 45 58 74 00 73 61 6e 73 20 63 6f 6d 6d 65 6e 74 61 69 72 65 21 b2 1b 5d 4b 00 00
00 00 49 45 4e 44 ae 42 60 82
```

address	name	type	size	data	description
00000000.0	id	Bytes	00000008.0	"\x89PNG\r\n\x1a\n"	PNG identifier ('\x89PNG\r\n\x1A\n')
00000008.0	header/	Chunk	00000025.0		Header: 538x190 pixels and 32 bits/pixel
00000021.0	data[0]/	Chunk	00013977.0		Image data
000036ba.0	text[0]/	Chunk	00000030.0		Text: "sans commentaire!"
000036d8.0	end/	Chunk	00000012.0		End

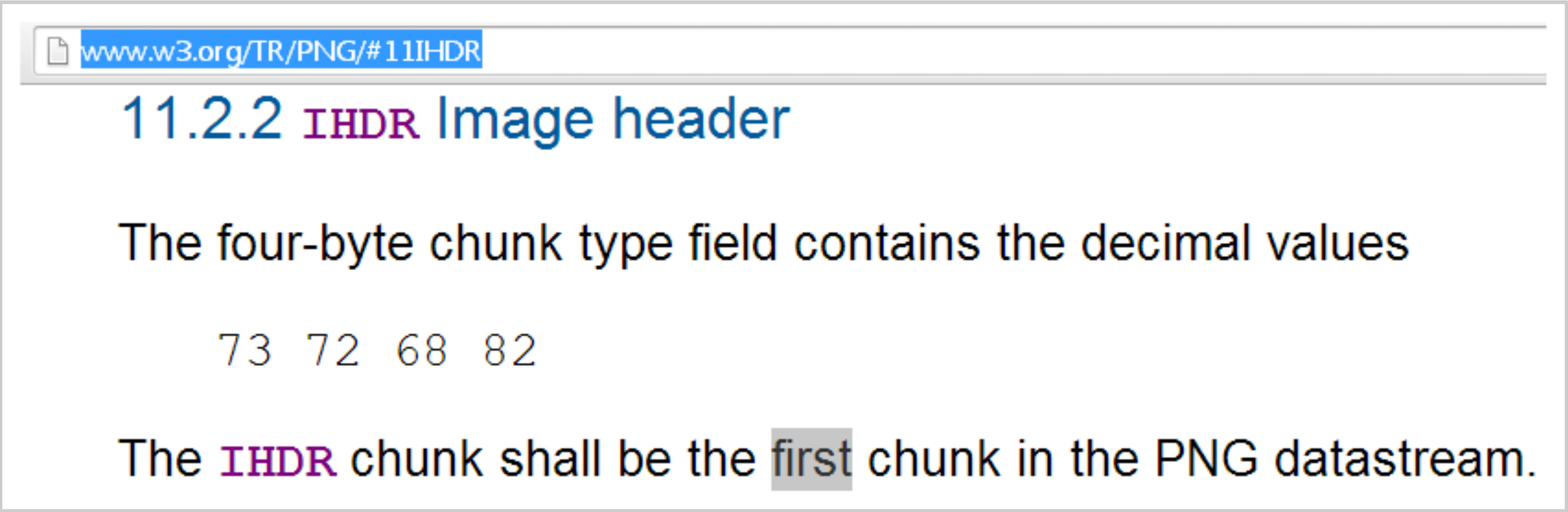
The chunk position doesn’t matter.



```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 02 1a 00 00 00 be 08 06 00 00 00 73 ab a6 f7 00
00 00 0b 62 69 6e 67 65 78 74 72 61 20 63 68 75 6e 6b 49 51 eb e3 00 00 36 8d 49 44 41 54 58 c3 ec d9
cb 7a d3 66 02 c6 71 3a 9d 43 db 95 9f 67 3a 09 98 10 4c e7 06 bc 9e 92 a0 70 ca 6e 6a 42 c8 81 43 10
dd b5 a5 c5 84 ce 5e 77 e0 1b 68 10 39 c1 ec 7c 05 45 a1 37 e0 f5 84 83 92 6d a1 c8 77 f0 cd fb c9 96
2d d9 92 ad 93 63 5b 7e df e7 f9 af ba aa ec ef cb 0f f9 94 10 e2 14 63 8c 31 c6 d8 20 e2 43 60 8c 31
c6 d8 60 a1 31 69 9b 79 6c 29 b2 b3 65 4b b3 7b 64 55 91 21 cb 3f b2 6a 48 74 76 e6 27 a7 8f 9e 4e ff
f8 d1 44 86 ab 0a d2 a6 1b 29 b2 53 1c c7 71 1c 37 a9 cb 32 34 ce 3d b6 0a a8 84 34 c0 c2 40 26 12 4e
40 46 a3 47 ed f2 01 f5 80 86 6f d3 fe 19 d3 0f ff d0 91 86 14 94 e3 37 90 e3 38 8e 23 34 c6 64 b3 9b
f5 c2 ec a6 55 06 2c aa c8 44 c2 69 c6 a7 16 34 42 60 23 25 68 08 e0 a2 33 b3 89 8f f2 d4 c3 0f 45 7e
23 39 8e e3 38 42 63 84 76 7e b3 5e 02 30 2a c8 44 42 e6 06 46 2f 6c 44 81 46 1b 1b a9 43 a3 15 a0 21
```

address	name	type	size	data	description
	../				
00000021.0	size	UInt32	00000004.0	11	Size
00000025.0	tag	FixedString<ASCII>	00000004.0	"bing"	Tag
00000029.0	content	RawBytes	00000011.0	"extra chunk"	Data
00000034.0	crc32	UInt32	00000004.0	0x4951ebe3	CRC32

Adding a completely custom `bing` chunk.

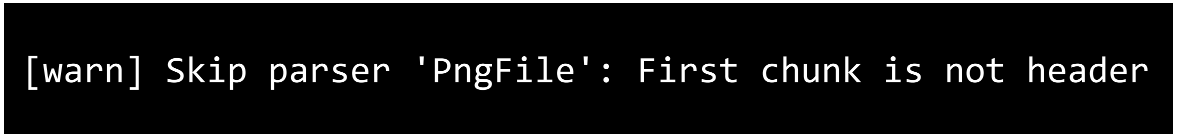


11.2.2 IHDR Image header

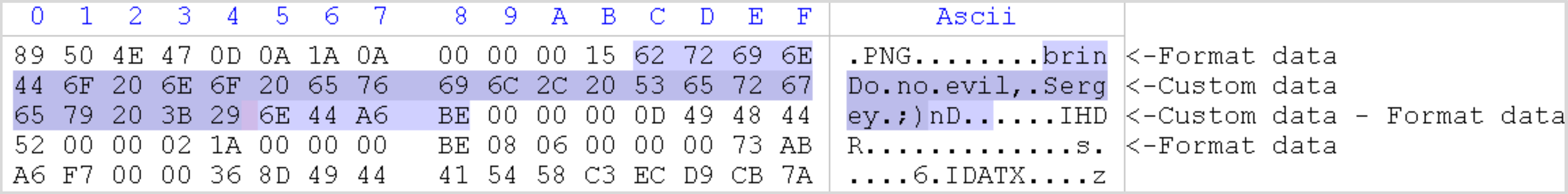
The four-byte chunk type field contains the decimal values

73 72 68 82

The IHDR chunk shall be the first chunk in the PNG datastream.



```
[warn] Skip parser 'PngFile': First chunk is not header
```



```
 0  1  2  3  4  5  6  7    8  9  A  B  C  D  E  F        Ascii
89 50 4E 47 0D 0A 1A 0A   00 00 00 15 62 72 69 6E   .PNG........brin  <-Format data
44 6F 20 6E 6F 20 65 76   69 6C 2C 20 53 65 72 67   Do.no.evil,.Serg  <-Custom data
65 79 20 3B 29 6E 44 A6   BE 00 00 00 0D 49 48 44   ey.;)nD.....IHD   <-Custom data - Format data
52 00 00 02 1A 00 00 00   BE 08 06 00 00 00 73 AB   R.............s.  <-Format data
A6 F7 00 00 36 8D 49 44   41 54 58 C3 EC D9 CB 7A   ....6.IDATX....z
```

The header chunk *should* be the first one.
In practice, it doesn't matter (except for Hachoir)

Recap

Adding custom chunks:

- lowercase *type*
- chunk order doesn’t matter much

⇒ we can add any extra data
in a custom chunk

add a custom chunk to *cover* encrypted data

⇒ it will be ignored
⇒ the encrypted file will be valid!

Status

We control after encryption:

- the first block
  - the signature (8 bytes)
  - 8 extra bytes
    - enough to declare a chunk
      (4 bytes of `size` + 4 bytes of `type`)
- the chunks of the target
  - by decrypting them in advance

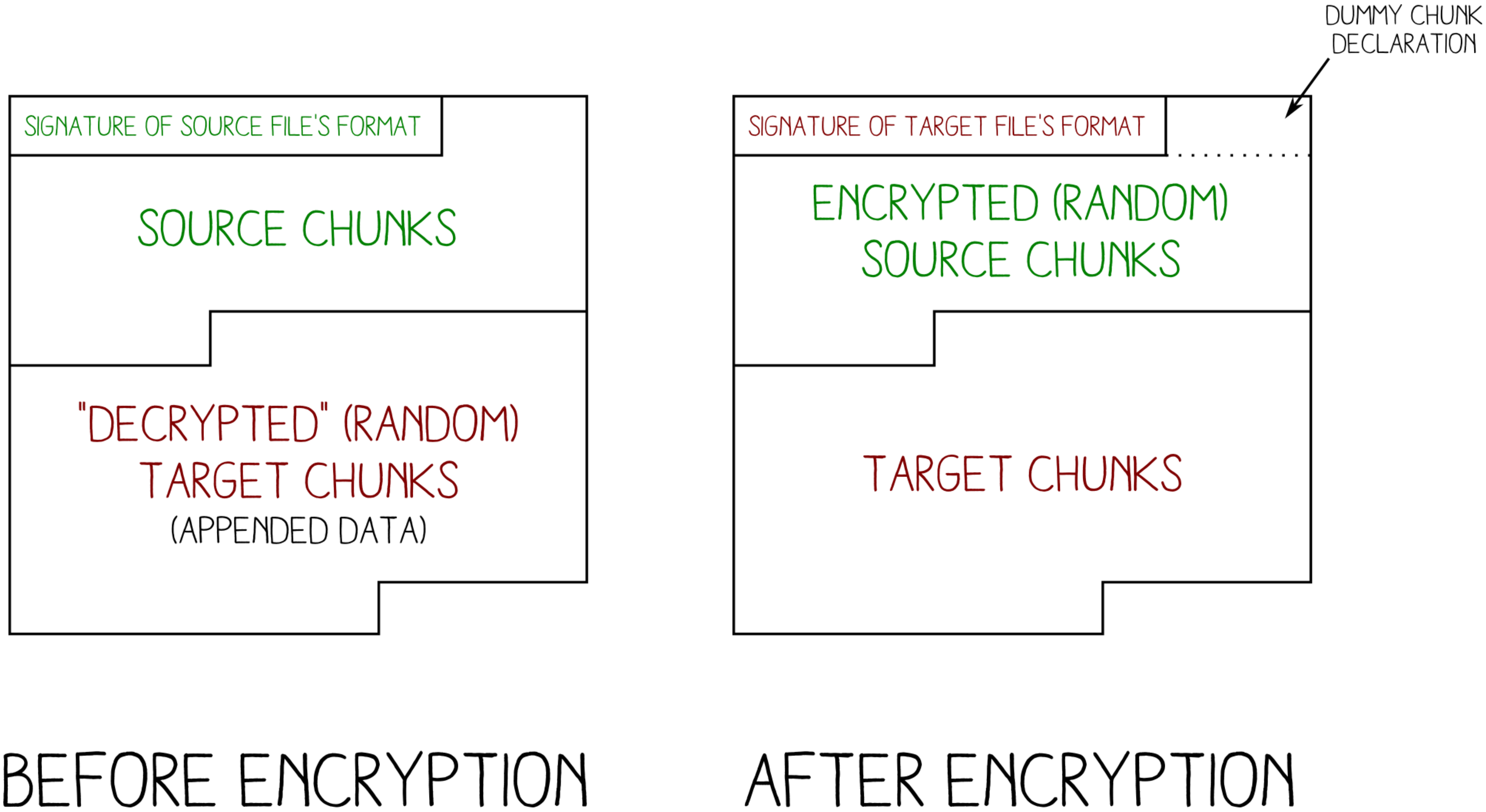
“AngeCryption”

with 2 files ***S***ource and ***T***arget,

- create a ***R***esult file

R shows

- S, initially
- T, after *AES-CBC(key, IV)* encryption



SIGNATURE OF SOURCE FILE'S FORMAT

SOURCE CHUNKS

"DECRYPTED" (RANDOM) TARGET CHUNKS

(APPENDED DATA)

BEFORE ENCRYPTION

DUMMY CHUNK DECLARATION

SIGNATURE OF TARGET FILE'S FORMAT

ENCRYPTED (RANDOM) SOURCE CHUNKS

TARGET CHUNKS

AFTER ENCRYPTION

File layout

Step by step

Initial data

We define the key, and the S and T files.

key	AngeCryptionKey!
S	
T	

Initial checks

- S is a PNG
  - the PNG format tolerates appended data
- T is a PNG
  - it allows custom chunks
    (at the beginning of the file, right after the signature)
- S fits in a single chunk
  - its size can be encoded in 4 bytes
- AES-128 has a 16 bytes block size
  - big enough to declare a chunk after the signature

Determine the first cipher block

- R starts with P1, from S
- once encrypted, R starts with:
  a. an 8 byte PNG signature
  b. a custom chunk
     - that covers all the chunks from S
       1. S is `14022` bytes, so that’s `14016` bytes of chunks
       2. `14016` is encoded `000036c0`
     - with a custom type: **rmll**

       lowercase ⇒ ancillary ⇒ ignored

First cipher block of R, C1:

```
89  P  N  G \r \n 1A \n   00 00 36 C0    r  m  l  l
Signature --------------   Length -----    Type ------
```

First plaintext / cipher blocks

First block of R, P1:

```
89  P  N  G \r \n 1A \n    00 00 00 0D    I  H  D  R
Signature -------------   Length -----   Type ------
```

First block of encrypted R, C1:

```
89  P  N  G \r \n 1A \n    00 00 36 C0    r  m  l  l
Signature -------------   Length -----   Type ------
```

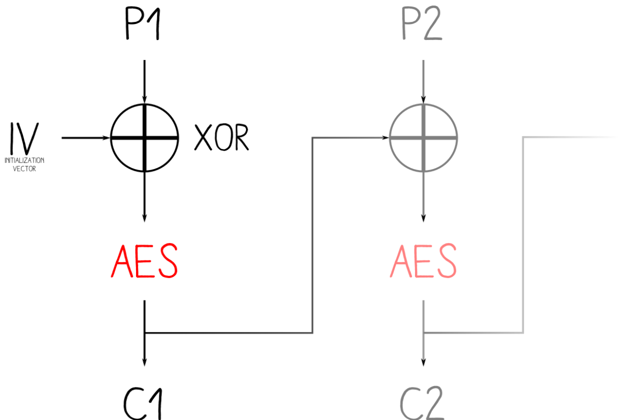
Determine the IV

We have the P1 & C1 blocks, and the key

1. Decrypt C1
2. XOR with P1

We get the IV that will encrypt P1 into C1:

```
78 D0 02 81 6B A7 C3 DE 88 DE 56 8F 6A 59 1D 06
```



Craft R

The IV is determined.

- *Pad* S to the next 16 bytes alignment
- Encrypt via AES-CBC with our parameters

→ with this IV, S will start with:
(after encryption)

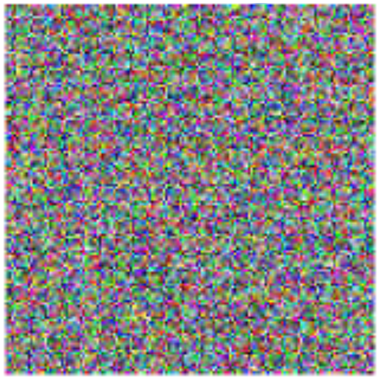
1. a signature
2. a `rmll` chunk (covering the rest of S)

Adjust the custom chunk

1. Chunks end with a CRC32
   - calculate it (using the encrypted data)
2. Append T’s plaintext chunks
3. Decrypt the result
   - after padding

Result

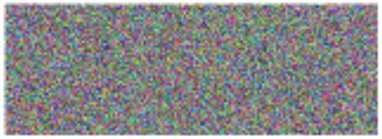
1. signature
2. S chunks
3. padding
4. T chunks (pre-decrypted)



```
0000:  89 50 4E 47-0D 0A 1A 0A-00 00 00 0D-49 48 44 52  ëPNG        IHDR
0010:  00 00 02 1A-00 00 00 BE-08 06 00 00-00 73 AB A6         +     s½ª
0020:  F7 00 00 36-8D 49 44 41-54 58 C3 EC-D9 CB 7A D3  ˜  6ìIDATX+8+-z+
0030:  66 02 C6 71-3A 9D 43 DB-95 9F 67 3A-09 98 10 4C  f ¦q:¥C¦òƒg: ÿ L
...
36A0:  F5 2D D0 30-33 33 33 BB-C4 3C 04 33-33 33 BB D8  )--0333+-< 333++
36B0:  FE 07 A4 AD-F2 BC 37 7B-32 76 00 00-00 00 49 45  ¦ ñ¡=+7{2v    IE
36C0:  4E 44 AE 42-60 82 00 00-00 00 00 00-00 00 00 00  ND«B`é..........
36D0:  43 F7 62 F2-4C 6A 07 4D-03 41 82 84-3C D3 F4 39  C˜b=Lj M Aéä<+(9
36E0:  FC 27 90 6B-82 71 C8 34-3E 48 4D C1-4C 2A BB 96  n'Ékéq+4>HM-L*+û
36F0:  3C 97 01 67-FE B3 E4 03-E9 09 B2 C3-7E 54 B7 23  <ù g¦¦S T ¦+~T+#
3700:  57 37 3F 1E-DF 67 B3 E8-60 B3 EC A6-CA 51 61 11  W7? ¯g¦F`¦8ª-Qa
...
5CE0:  CC 22 8A A0-EC 19 8C DD-26 79 03 29-03 90 93 F1  ¦"èá8 î¦&y ) Éô±
5CF0:  41 CE 4F DB-C0 70 A5 74-D0 74 B7 2E-06 9B 48 7C  A+O¦+pÑt-t+. ¢H|
5D00:  2F A6 D1 ED-57 FB 88 67-D1 B0 10 4C-1C 6E CB 15  /ª-fWvêg-¦ L n-
```

Encrypted result

1. signature
2. custom chunk
   a. CRC32
3. T chunks
4. padding



```
0000:  89 50 4E 47-0D 0A 1A 0A-00 00 36 C0-72 6D 6C 6C  ëPNG      6+rmll
0010:  9A 3E 30 1C-F1 D6 E1 41-B7 38 DB A1-5A 71 57 8F  Ü>0 ±+ßA+8¦íZqWÅ
0020:  6E 49 A0 D5-76 4C 33 7D-9B CA 44 B8-72 27 48 D9  nIá+vL3}¢-D+r'H+
0030:  64 20 A6 7F-38 D8 89 4A-9F 5F 92 45-17 5D 70 BA  d ª 8+ëJƒ_ÆE ]p¦
...
36A0:  4D 1E 79 E7-9E F5 81 AC-0C 4C 3B 03-75 43 2B 15  M ytP)ü¼ L; uC+
36B0:  B6 9F F4 32-E8 3C 02 67-96 DA 7B 1D-A8 E5 1E BF  ¦ƒ(2F< gû+{ ¿s +
36C0:  D1 04 25 DF-E5 92 E3 62-30 9A F6 08-60 57 BC 5B  - %¯sÆpb0Ü÷ `W+[
36D0:  98 38 F0 D6-00 00 00 0D-49 48 44 52-00 00 00 86  ÿ8=+    IHDR   å
36E0:  00 00 00 86-08 02 00 00-00 97 1B 65-C6 00 00 25     å     ù e¦  %
36F0:  FE 49 44 41-54 78 5E D4-C0 C1 0A 00-10 0C 00 50  ¦IDATx^++-     P
3700:  FF FF 6F CA-8D B8 A8 95-92 1C 56 0E-36 9B F9 0E    o-ì+¿òÆ V 6¢·
...
5CE0:  EE 4B 05 D4-46 49 B3 66-30 ED 6E BF-E7 23 7B C9  eK +FI¦f0fn+t#{+
5CF0:  C8 D7 51 F8-99 B7 9C 00-00 00 00 49-45 4E 44 AE  ++Q°Ö+£    IEND«
5D00:  42 60 82 00-00 00 00 00-00 00 00 00-00 00 00 00  B`é.............
```

Generalized case

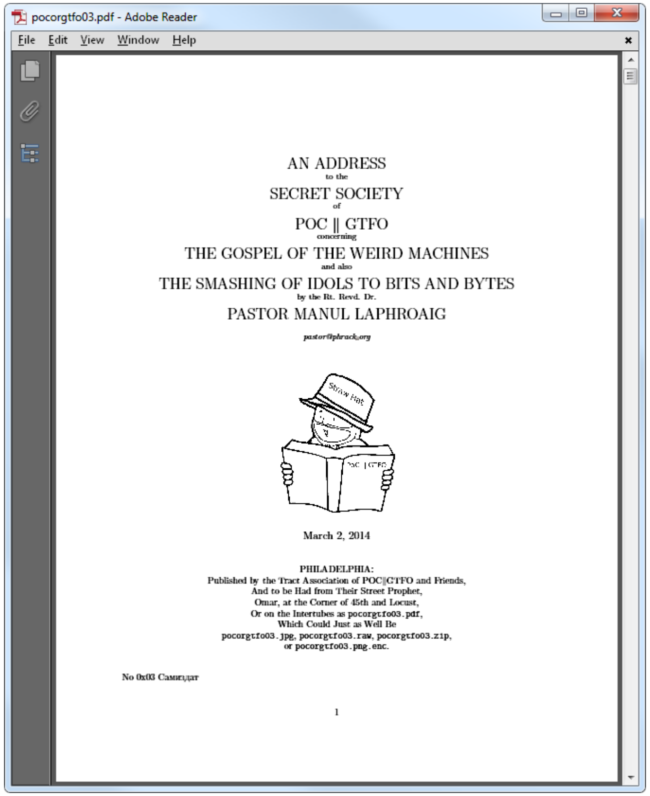
The only requirements:

- The source format tolerates appended data
- The target format can fit a signature and chunk declaration in a single cipher block
- S fits in a single target format chunk

We can use other algorithms,

both ways (encryption or decryption)

with various file formats (even in the same file)



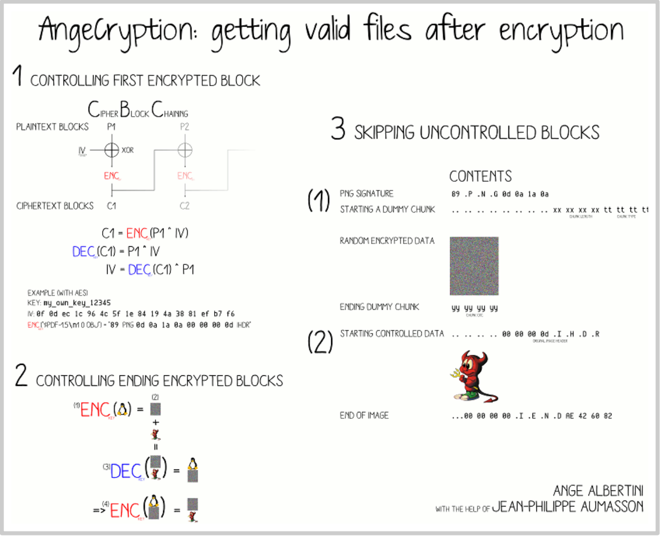
AN ADDRESS
to the
SECRET SOCIETY
of
POC || GTFO
concerning
THE GOSPEL OF THE WEIRD MACHINES
and also
THE SMASHING OF IDOLS TO BITS AND BYTES
by the Rt. Revd. Dr.
PASTOR MANUL LAPHROAIG
pastor@phrack.org

March 2, 2014

PHILADELPHIA:
Published by the Tract Association of POC||GTFO and Friends,
And to be Had from Their Street Prophet,
Omar, at the Corner of 45th and Locust,
Or on the Intertubes as pocorgtfo03.pdf,
Which Could Just as Well Be
pocorgtfo03.jpg, pocorgtfo03.raw, pocorgtfo03.zip,
or pocorgtfo03.png.enc.

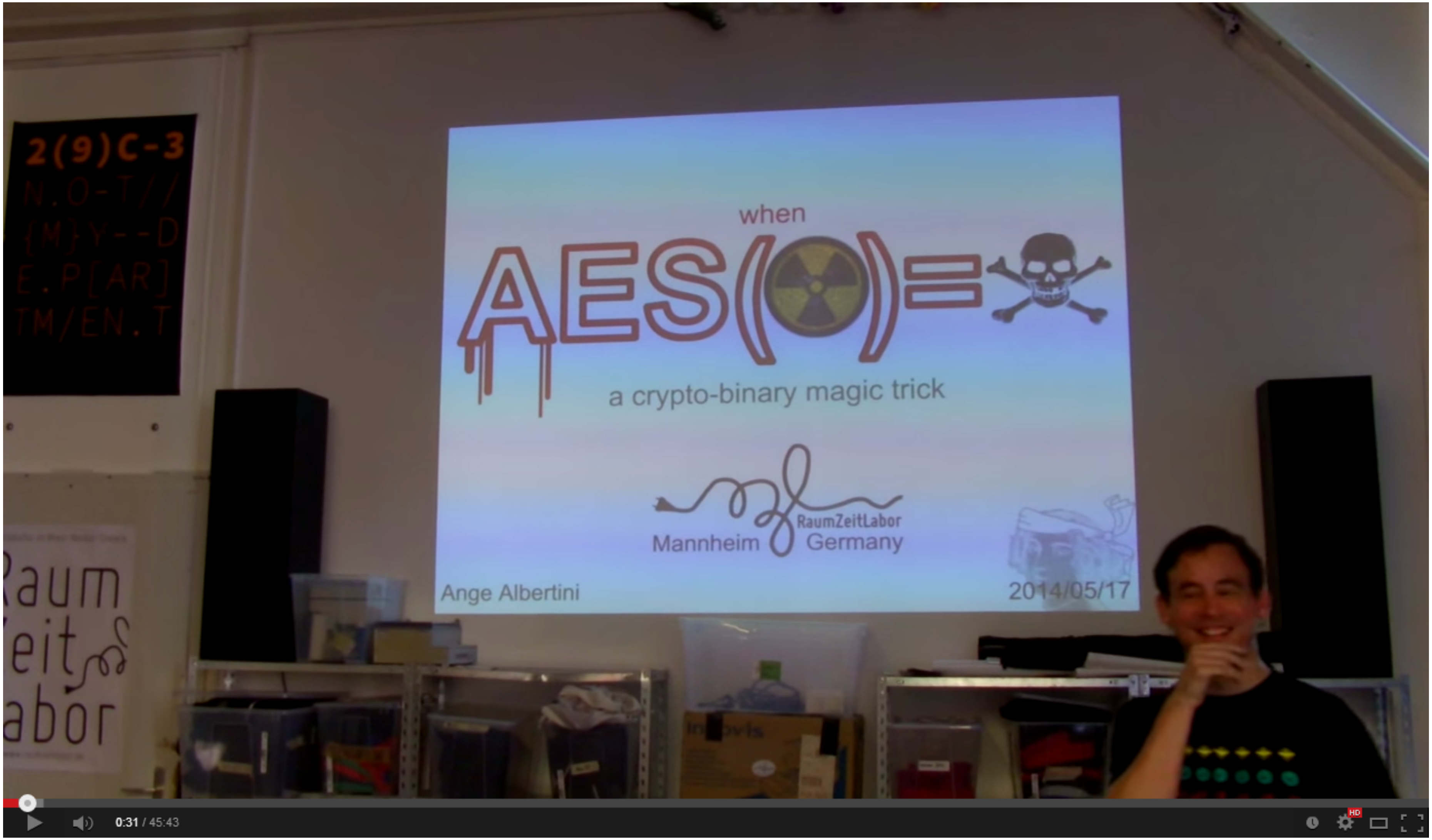
No 0x03 Самиздат

1



Technical Note: This file, `pocorgtfo03.pdf`, complies with the PDF, JPEG, and ZIP file formats. When encrypted with AES in CBC mode with an IV of `5B F0 15 E2 04 8C E3 D3 8C 3A 97 E7 8B 79 5B C1` and a key of "`Manul Laphroaig!`", it becomes a valid PNG file. Treated as single-channel raw audio, 16-bit signed little-endian integer, at a sample rate of 22,050 Hz, it contains a 2400 baud AFSK transmission.

PoC||GTFO 0x3 is a PDF that you can encrypt into a PNG
(and it shows its own IV)



when AES(☢)=☠

a crypto-binary magic trick

RaumZeitLabor

Mannheim Germany

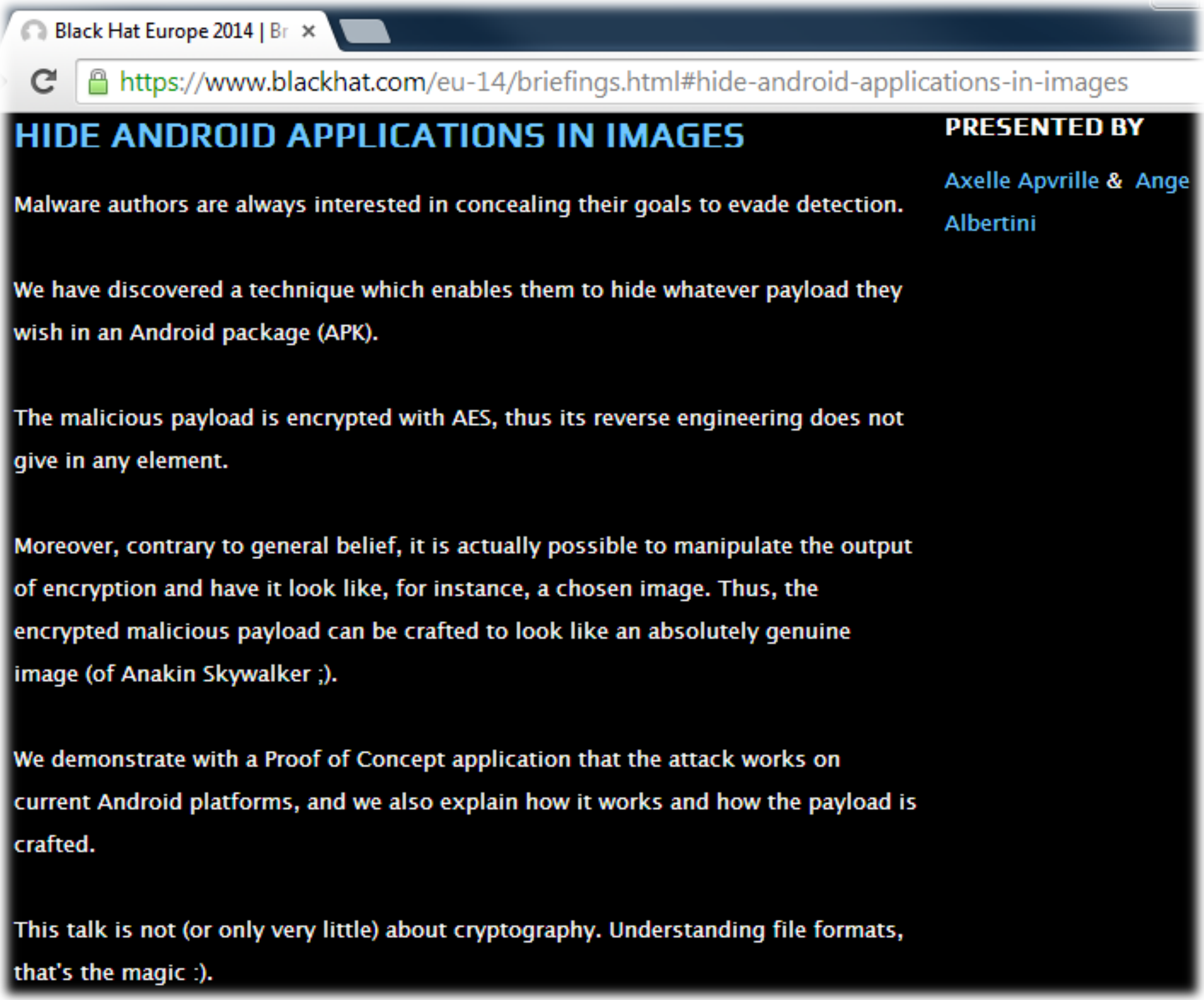
Ange Albertini

2014/05/17

For more information (PDF, JPG, GynCryption, PiP…):

https://speakerdeck.com/ange/when-aes-equals-episode-v

https://www.youtube.com/watch?v=wbHkVZfCNuE



Black Hat Europe 2014 | Br
https://www.blackhat.com/eu-14/briefings.html#hide-android-applications-in-images

HIDE ANDROID APPLICATIONS IN IMAGES

PRESENTED BY

Axelle Apvrille & Ange Albertini

Malware authors are always interested in concealing their goals to evade detection.

We have discovered a technique which enables them to hide whatever payload they wish in an Android package (APK).

The malicious payload is encrypted with AES, thus its reverse engineering does not give in any element.

Moreover, contrary to general belief, it is actually possible to manipulate the output of encryption and have it look like, for instance, a chosen image. Thus, the encrypted malicious payload can be crafted to look like an absolutely genuine image (of Anakin Skywalker ;).

We demonstrate with a Proof of Concept application that the attack works on current Android platforms, and we also explain how it works and how the payload is crafted.

This talk is not (or only very little) about cryptography. Understanding file formats, that's the magic :).

Coming this fall...

Let’s play with TrueCrypt

TrueCrypt software

- Creates and manages a virtual storage volume
  - Encrypted
  - Transparent for the system

The volume is useless without the password.

```
ëPNG♪◙→◙     ♪IHDR
   ☻→     ╛◘♠     s½ª
≈   6ìIDATX├∞┘╦z╙
```

```
PK♥♦¶     ◘ î◄|>á#
SÇ≈♀   ♫∟   ↕    Cl
```

ZIP

```
 ╪ α ►JFIF ☺☺☺ ┤
 ┤    █ C ♠♦♣♠♣♦♠
♠♣♠••♠◘◙►◙◙○○◙¶♫
☼♀►↨¶↑↑↨¶▬→↔%▼→
←#∟▬  ,  #&')*)↓▼
-0-(0%()( █ C☺••
•◙◘◙!!◙◙!!(→▬→((((
 ((((((((((((((((
```

JPG

```
%PDF-1.3◙%─σ≥σδ°
≤á╨─╞◙4 0 obj◙<<
 /Length 5 0 R /
Filter /FlateDec
ode >>◙stream◙x☺
```

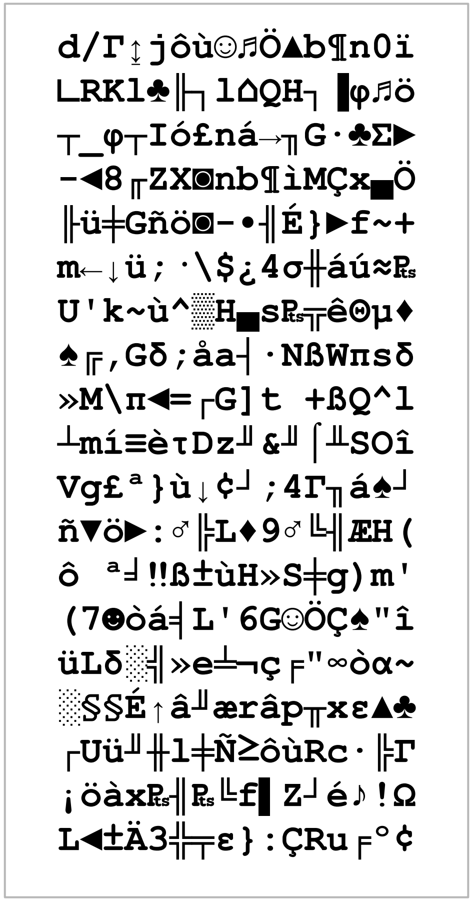
```
    ftypisom  ☻
isomiso2avc1mp41
   *freevideo se
rved by mod_h264
_streaming ▲¿<mo
ov   lmvhd     |%
░Ç|%░Ç  ♥Φ ),( ☺
```

```
MZÉ ♥    ♦
╕        @


            ◘☺
♫▼║♫ ┤○═!╕☺L═!Th
is program canno
t be run in DOS
mode.♪♪◙$
```

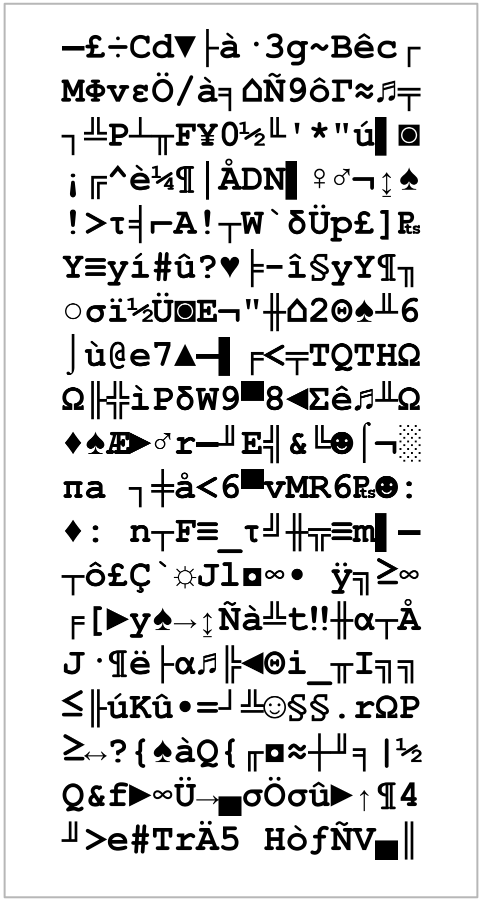
PE

```
⌂ELF☺☻☺♥
 ☻ ¶    ☺ ─:É    4
          4    ☻ (
        ☺      ►
 ►    ♠K¼ ♠K¼    ♣
 ☺      ☺   ·Φ►  ·Φ
►  ·Φ            ♠
 ☺   ╣E)lUPX!↓◄♪ä
```

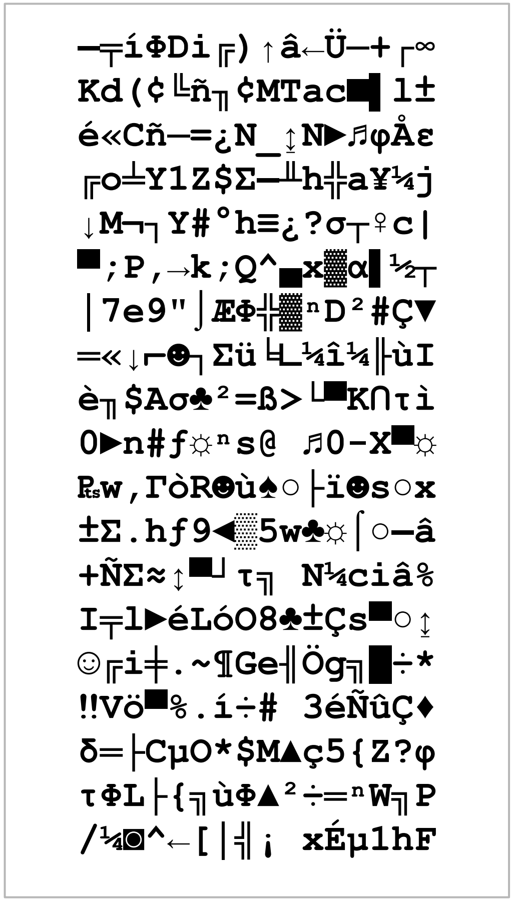
Standard file format headers



```
d/Γ↨jôù☺♫Ö▲b¶n0ï
└RKl♣╟┐ l⌂QH┐ ▌φ♫ö
┬_φ┬Ió£ná→╖G·♣Σ►
-◄8╓ZX◙nb¶ìMÇx▄Ö
╟ü╪Gñö◙-•╢É}►f~+
m←↓ü; ·\$¿4σ╫áú≈₧
U'k~ù^░H▄s₧╤êΘµ♦
♠╓,Gδ;åa┤ ·NßWπsδ
»M\π◄═┌G]t +ßQ^l
┴mí≡èτDz╜&╜⌠╨SOî
Vg£ª}ù↓¢┘;4Γ╖á♠┘
ñ▼ö►:♂╟L♦9♂╙╢ÆH(
ô ª╛‼ß±ùH»S╪g)m'
(7☻òá╡L'6G☺ÖÇ♠"î
üLδ░╣»e╧¬ç╒"∞òα~
░§§É↑â╜ærâp╥xε▲♣
┌Uü╜╫l╪Ñ≥ôùRc·╟Γ
¡öàx₧╢₧╚f▌Z┘é♪!Ω
L◄±Ä3╬╤ε}:ÇRu╒°¢
```



```
─£÷Cd▼├à ·3g~Bêc┌
MΦvεÖ/à╕⌂Ñ9ôΓ≈♫╤
┐╧P┴╥F¥0½╙'*"ú▌◙
¡╓^è¼¶│ÅDN▌♀♂¬↨♠
!>τ╡┌A!┬W`δÜp£]₧
Y≡yí#û?♥╞-î§yY¶╖
○σï½Ü◙E¬"╫⌂2Θ♠╨6
⌡ù@e7▲─▐ ╒<╤TQTHΩ
Ω╟╬ìPδW9▀8◄Σê♫╨Ω
♦♠Æ►♂r─╜E╣&╚☻⌠¬░
πa ┐╪å<6▀vMR6₧☻:
♦: n┬F≡_τ╝╫╥≡m▌─
┬ô£Ç`☼Jl◘∞• ÿ╖≥∞
╒[►y♠→↨Ñà╧t‼╫α┬Å
J·¶ë├α♫╟◄Θi_╥I╖╖
≤╟úKû•═┘╧☺§§.rΩP
≥↔?{♠àQ{╓◘≈┼╜╕|½
Q&f►∞Ü→▄σÖσû►↑¶4
╜>e#TrÄ5 HòƒÑV▄║
```



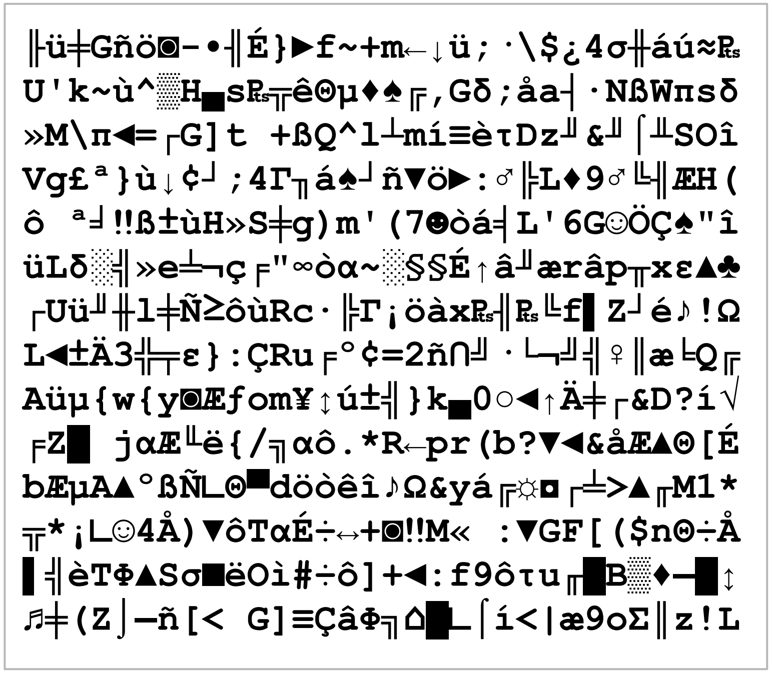
```
─╤íΦDi╓)↑â←Ü─+┌∞
Kd(¢╚ñ╖¢MTac▄▌ l±
é«Cñ─═¿N_↨N►♫φÅε
╓o╧Y1Z$Σ─╨h╫a¥¼j
↓M¬┐Y#°h≡¿?σ┬♀c|
▀;P,→k;Q^▄x▓α▌½┬
│7e9"⌡ÆΦ╬▓ⁿD²#Ç▼
═«↓┌☻┐Σü╘└¼î¼╟ùI
è╖$Aσ♣²═ß>└▀K∩τì
0►n#ƒ☼ⁿs@ ♫0-X▀☼
₧w,ΓòR☻ù♠○├ï☻s○x
±Σ.hƒ9◄░5w♣☼⌠○─â
+ÑΣ≈↕▀┘ τ╖ N¼ciâ%
I╤l►éLóO8♣±Çs▀○↨
☺╓i╪.~¶Ge╢Ög╖█÷*
‼Vö▀%.í÷# 3éÑûÇ♦
δ═├CµO*$M▲ç5{Z?φ
τΦL├{╖ùΦ▲²÷═ⁿW╖P
/¼◙^←[│╣¡ xɵ1hF
```

Headers of different TrueCrypt files.

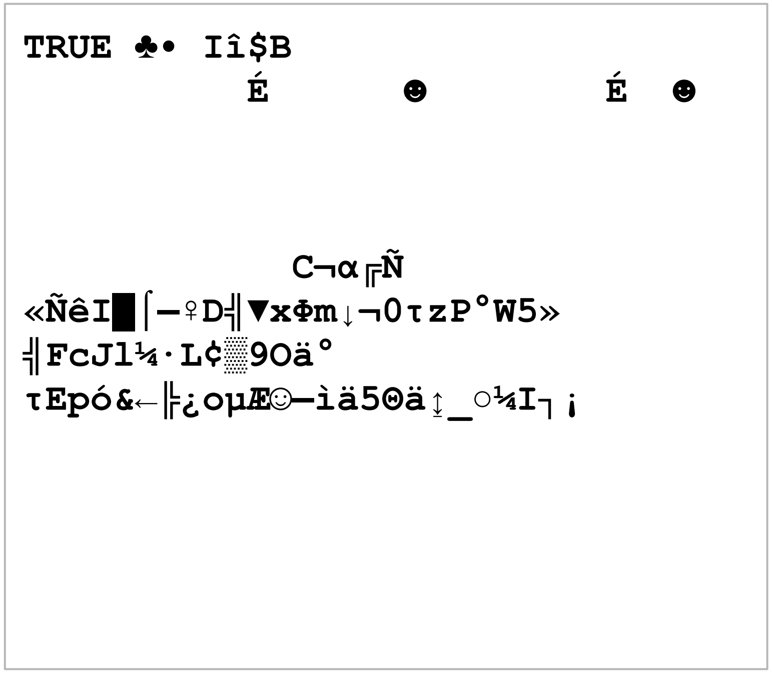
Random contents?

A file format designed **not** to be identified

- except if you have the password
- random appearance
  - you can deny it’s a TrueCrypt volume
- there **is** a header
  - but it’s encrypted



```
╟ü╪Gñö◙-•╢É}►f~+m←↓ü; ·\$¿4σ╫áú≈₧
U'k~ù^▒H▄s₧╤êΘµ♦♠╔,Gδ;åa┤ ·NßWπsδ
»M\π◄═┌G]t +ßQ^l┴mí≡èτDz╝&╜⌠╨SOî
Vg£ª}ù↓¢┘;4Γ╖á♠┘ñ▼ö►:♂╟L♦9♂╚╣ÆH(
ô ª╛‼ß±ùH»S╪g)m'(7☻òá╡L'6G☺ÖÇ♠"î
üLδ░╣»e╧¬ç╒"∞òα~░§§É↑â╜ærâp╥xε▲♣
┌Uü╜╫l╪Ñ≥ôùRc·╟Γ¡öàx₧╢₧╙f▌Z┘é♪!Ω
L◄±Ä3╬╤ε}:ÇRu╒°¢=2ñ∩╝ ·└¬╝╣♀║æ╘Q╔
Aüµ{w{y◙Æƒom¥↕ú±╣}k▄0○◄↑Ä╪┌&D?í√
╒Z█ jαÆ╙ë{/╗αô.*R←pr(b?▼◄&åÆ▲Θ[É
bÆµA▲°ßÑ╚Θ▀döòêî♪Ω&yá╔☼◘┌╧>▲╓M1*
╥*¡╚☺4Å)▼ôTαÉ÷↔+◙‼M« :▼GF[($nΘ÷Å
▌╣èTΦ▲Sσ■ëOì#÷ô]+◄:f9ôτu╓█B▒♦─█↕
♬╪(Z⌡─ñ[< G]≡ÇâΦ╖⌂█▄⌠í<|æ9oΣ║z!L
```



```
TRUE ♣• Iî$B
          É       ☻         É  ☻




            C¬α╔Ñ
«ÑêI█⌠─♀D╣▼xΦm↓¬0τzP°W5»
╣FcJl¼·L¢▒9Oä°
τEpó&←╟¿oµÆ☺─ìä5Θä↨_○¼I┐¡
```

A TrueCrypt header, before and after decryption.

How many files do you have that are 100% random?

it’s not ***so*** stealthy

Potential volumes detection

- no known header
- “big size”
- size rounded to 512
- random content from the start
  - very high *entropy*

Just a password?

If encryption only depends on the password, TrueCrypt is vulnerable to [*] table attacks.

[*] pre-computed tables (to make faster attacks)

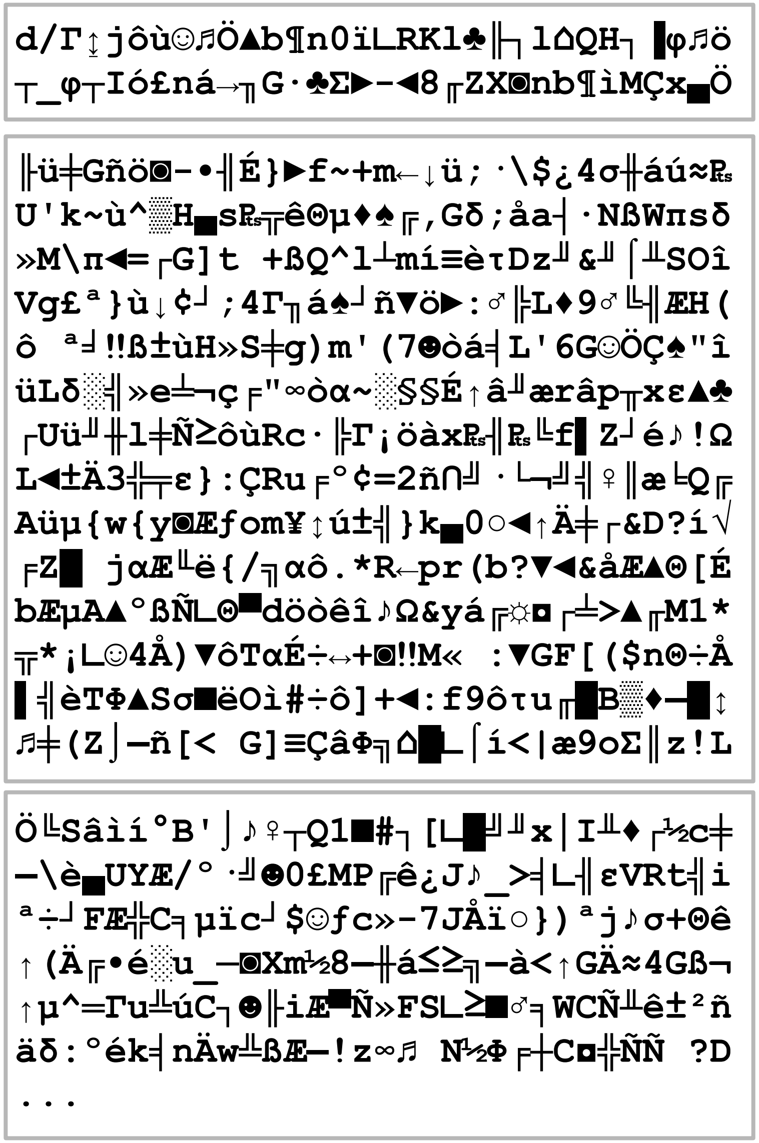
Salt

The file starts with 64 bytes of salt:

- random data
- combined with the password
- used to decrypt the header

⇒ no possible pre-computing

⇒ rainbow tables are useless



Salt
(to decrypt the header)

Header

- crypted with salt and password
- contains the key used to decrypt the volume

Volume

- encrypted with the key in the header

Structure of a TrueCrypt volume

If we modify the salt, we just have to to re-crypt the header

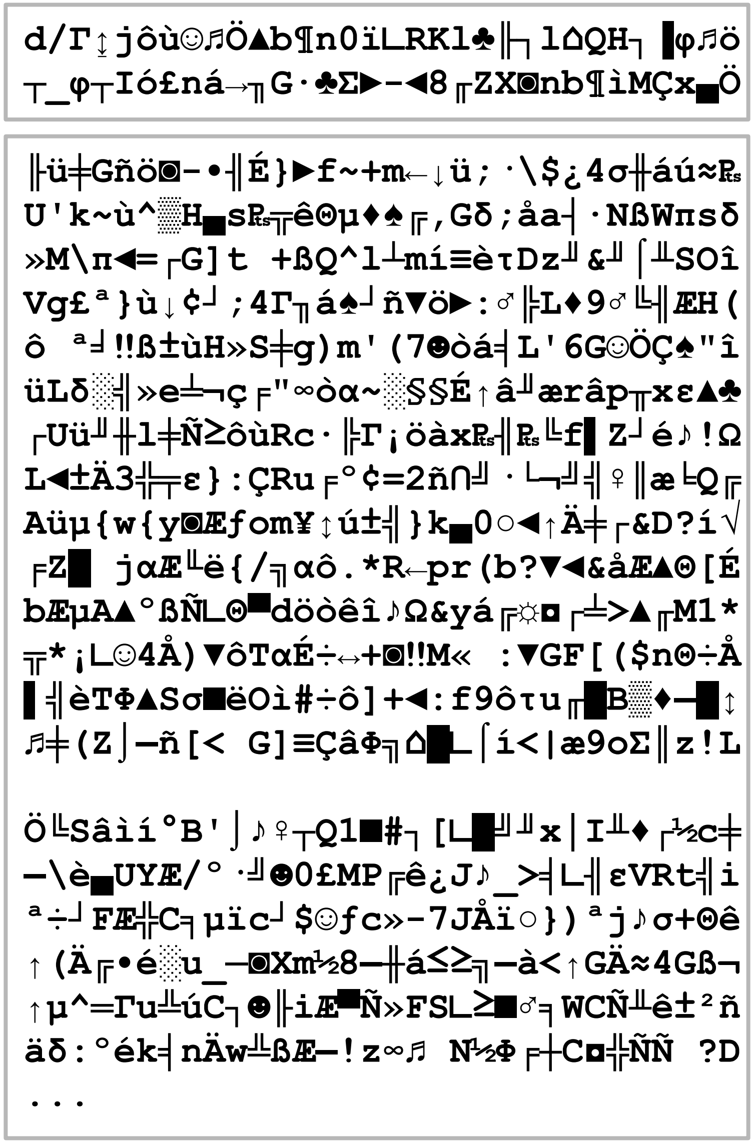
no need to change the volume itself
(the volume key hasn’t changed)

Idea:
Integrate a TrueCrypt volume into another file

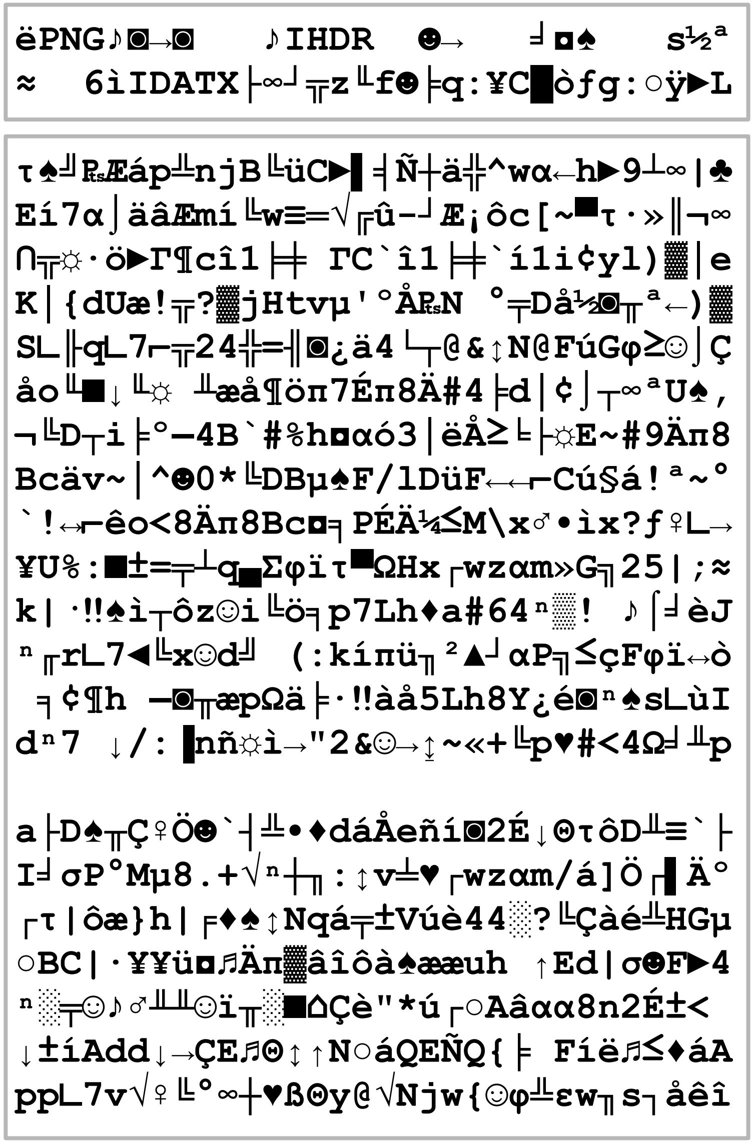
- stealthier
- both formats stay valid

Strategy

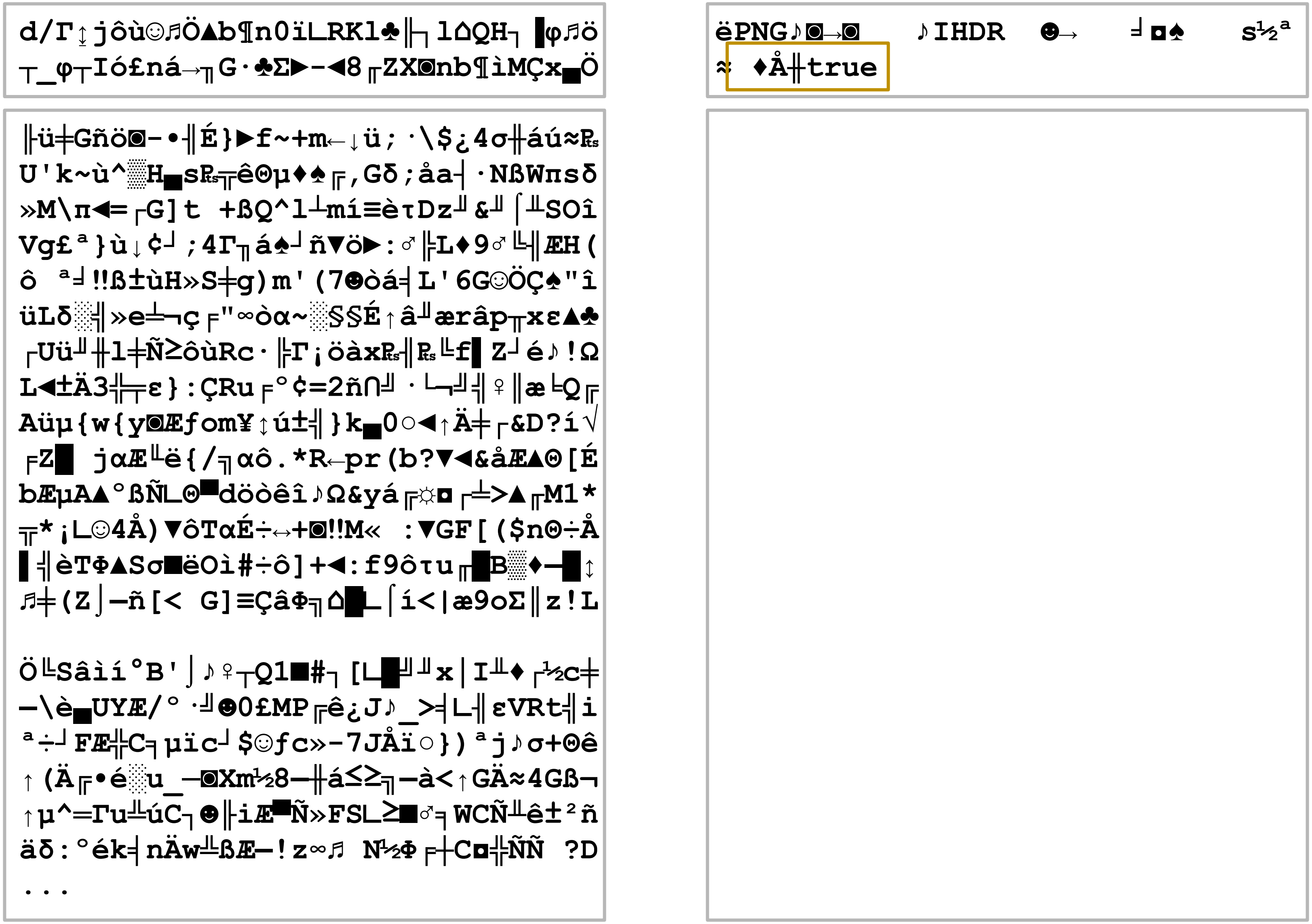
1. Modify the host to make free space near the beginning
   - create a custom chunk to contain the volume
2. Copy the header and the volume's content
   - the decrypted header hasn't changed,
     and the volume hasn't either
3. Decrypt the header
   - with the initial salt
4. Re-crypt the header
   - with the salt from the start of the host
5. Adjust the CRC of the chunk
   - optional, as the chunk is ancillary



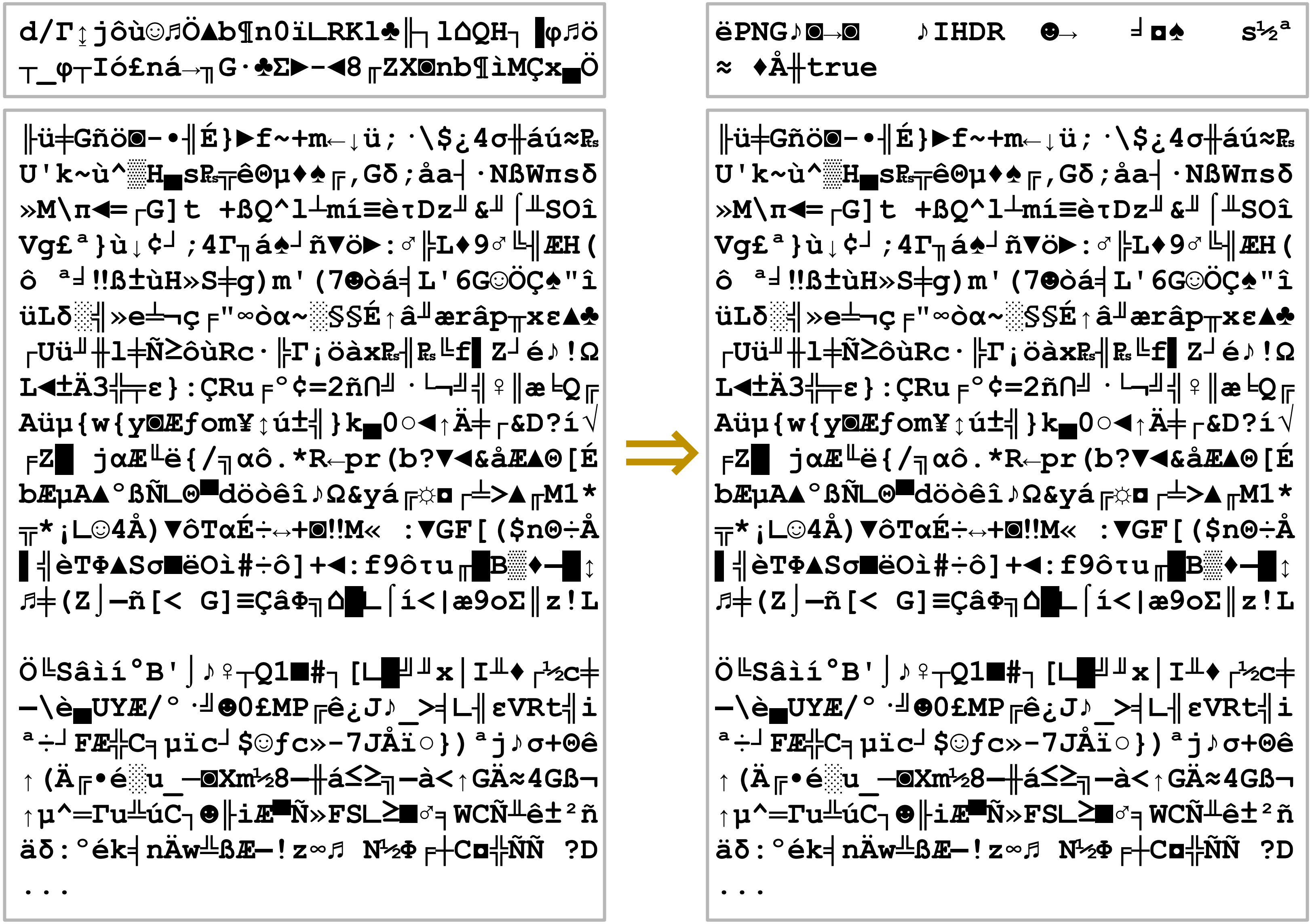
TrueCrypt volume



Image



Create free space in the file to host the volume.



Copy the volume in the created space.

```
d/Γ↨jôù☺♬Ö▲b¶n0ïLRKl♣╟┬ lΔQH┐▐φ♬ö
┬_φ┬Ió£ná→╖G·♣Σ►-◄8╓ZX◙nb¶ìMÇx▄Ö
```

```
ëPNG♪◙→◙    ♪IHDR   ☻→     ╛◘♠    s½ª
≈ ♦Å╫true
```

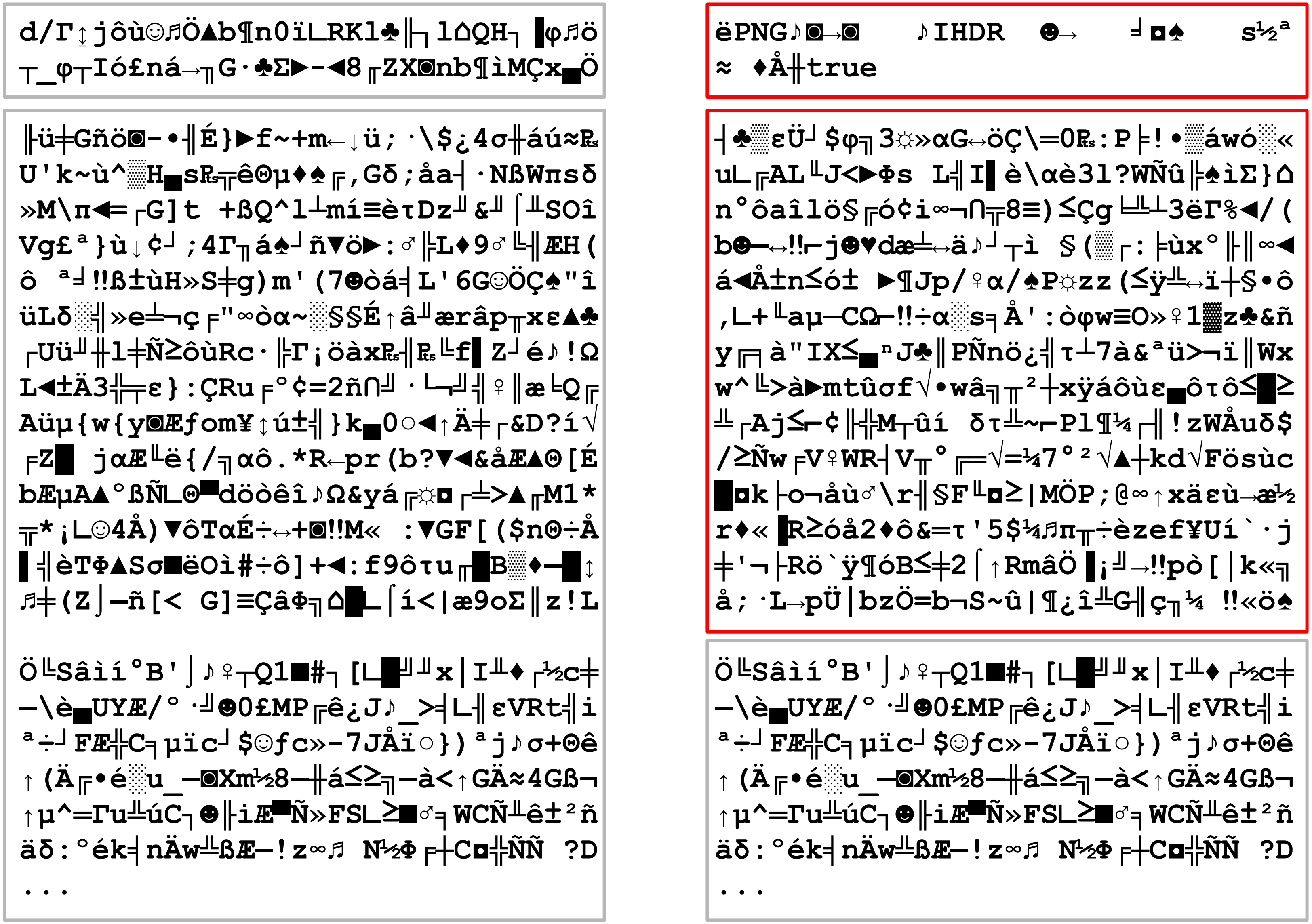
```
╟ü╪Gñö◙-•╢É}►f~+m←↓ü; ·\$¿4σ╫áú≈₧
U'k~ù^▒H▄s₧╦êΘµ♦♠╓,Gδ;åa┤ ·NßWπsδ
»M\π◄═╒G]t +ßQ^l┴mí≡èτDz╜&╜⌠╨SOî
Vg£ª}ù↓¢┘ ;4Γ╖á♠┘ñ▼ö►:♂╠L♦9♂╚╢ÆH(
ô ª╛‼ß±ùH»S╪g)m'(7☻òá╡L'6G☺ÖÇ♠"î
üLδ░╢»e╧¬ç╒"∞òα~░§§É↑â╜ærâp╥xε▲♣
┌Uü╜╫l╪Ñ≥ôùRc·╟Γ¡öàx₧╢₧╙f▌Z┘é♪!Ω
L◄±Ä3╬╤ε}:ÇRu╒°¢=2ñ∩╝ ·└¬╜╢♀║æ╘Q╓
Aüµ{w{y◙Æfom¥↨ú±╣}k▄0○◄↑Ä╪┌&D?í√
╒Z▌ jαÆ╙ë{/╗αô.*R←pr(b?▼◄&åÆ▲Θ[É
bÆµA▲°ßÑ└Θ▀döòêî♪Ω&yá╓☼◘ ┌╧>▲╓M1*
╥*¡L☺4Å)▼ôTαÉ÷↔+◙‼M« :▼GF[($nΘ÷Å
▌╣èTΦ▲Sσ■ëOì#÷ô]+◄:f9ôτu╓█B▒♦─█↕
♬╪(Z⌡─ñ[< G]≡ÇâΦ╖⌂█▙⌠í<|æ9oΣ║z!L
```

```
TRUE ♣•  IîSB
             É        ☻          É   ☻





C¬α╓Ñ«ÑêI█⌠─♀D╢▼xΦm↓¬0τzP°W5»╣Fc
Jl¼·L¢▒9Oä°τEpó&←╠¿oµÆ☺─ìä5Θä↨_○
¼I┐¡
```

```
Ö╙Sâìí°B'⌡♪♀┬Q1■#┐[L█╜╜x|I╨♦┌½c╪
─\è▄UYÆ/° ·╜☻0£MP╓ê¿J♪_>╡L╢εVRt╢i
ª÷┘ FÆ╬C╕µïc┘ $☺fc»-7JÅï○}) ªj♪σ+Θê
↑(Ä╓•é░u_─◙Xm½8─╫á≤≥╗─à<↑GÄ≈4Gß¬
↑µ^=Γu╧úC┐☻╟iÆ▀Ñ»FSL≥■♂╕WCÑ╨ê±²ñ
äδ:°ék╡nÄw╧ßÆ─!z∞♬ N½Φ╒┼C◘╬ÑÑ ?D
...
```

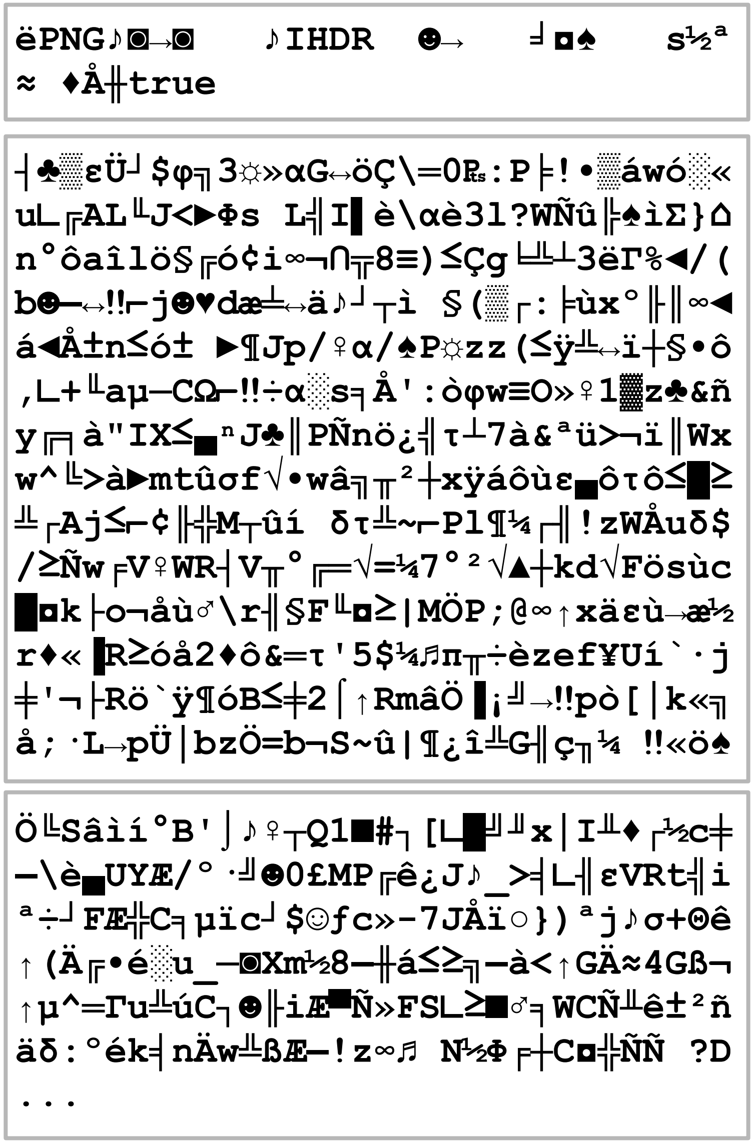
```
Ö╙Sâìí°B'⌡♪♀┬Q1■#┐[L█╜╜x|I╨♦┌½c╪
─\è▄UYÆ/° ·╜☻0£MP╓ê¿J♪_>╡L╢εVRt╢i
ª÷┘ FÆ╬C╕µïc┘ $☺fc»-7JÅï○}) ªj♪σ+Θê
↑(Ä╓•é░u_─◙Xm½8─╫á≤≥╗─à<↑GÄ≈4Gß¬
↑µ^=Γu╧úC┐☻╟iÆ▀Ñ»FSL≥■♂╕WCÑ╨ê±²ñ
äδ:°ék╡nÄw╧ßÆ─!z∞♬ N½Φ╒┼C◘╬ÑÑ ?D
...
```

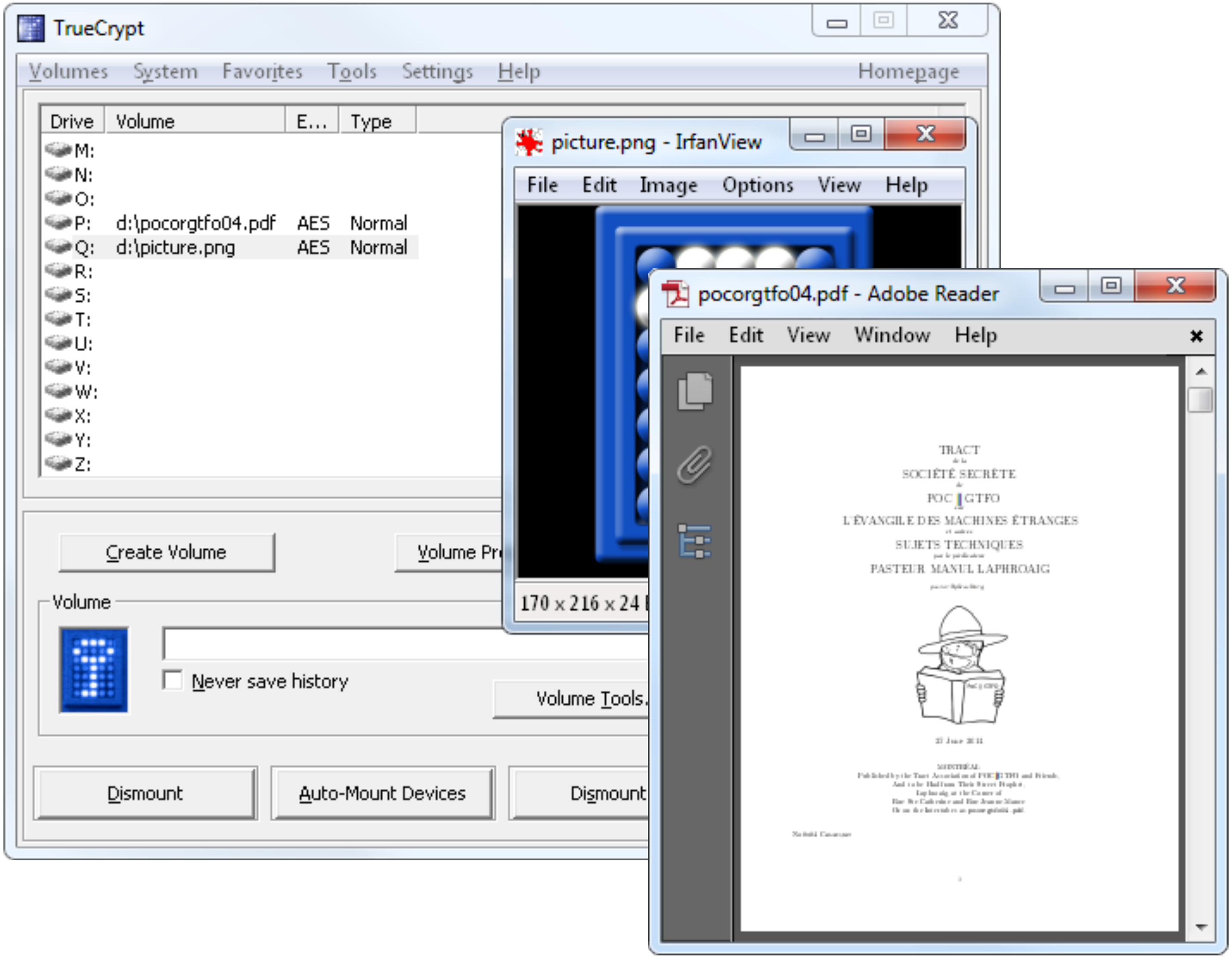
Decrypt the header with the volume’s salt.



Encrypt the header with the salt from the host.

TrueCrypt	PNG
Sal	Signature + header Chunk declaration
Header	Chunk data
Volume’s content	
End of volume	Chunk end
Image (ignored) ✓	Original chunks ✓





Drive	Volume	E...	Type
M:			
N:			
O:			
P:	d:\pocorgtfo04.pdf	AES	Normal
Q:	d:\picture.png	AES	Normal
R:			
S:			
T:			
U:			
V:			
W:			
X:			
Y:			
Z:			

TRACT
de la
SOCIÉTÉ SECRÈTE
de
POC‖GTFO
sur
L'ÉVANGILE DES MACHINES ÉTRANGES
et autres
SUJETS TECHNIQUES
par le prédicateur
PASTEUR MANUL LAPHROAIG
[illegible]

23 June 2014

MONTRÉAL
Published by the Tract Association of POC‖GTFO and Friends,
And to be Had from Their Street Prophet,
Laphroaig at the Corner of
Rue Ste Catherine and Rue Jeanne Mance
Or on the Internets as pocorgtfo04.pdf

[illegible]

TrueCrypt volumes in standard files
(still useable and modifiable)

Conclusion 1/2

- We can add extra data in a standard binary file
- This data can be:
  - another standard file, after en/decryption
  - a TrueCrypt volume

Conclusion 2/2

- No need to understand everything to have fun with crypto
- Better progress step by step
  - ask an expert
  - hard to debug
- Encrypted doesn’t mean random
- examples: http://bit.ly/1n63yKP

  (http://corkami.googlecode.com/svn/trunk/src/angecryption/rmll)

Acknowledgments

@veorq @doegox @iamreddave

@miaubiz @travisgoodspeed @sergeybratus
@cynicalsecurity @rantyben @thegrugq
@skier_t @jvanegue @kaepora @munin
@joernchen @andreasdotorg @tabascoeye
@cryptax @pinkflawd @push_pnx @gynvael
@rfidiot @cbrocas @kennwhite...

@angealbertini

corkami.com