when AES(☢)=☠

a crypto-binary magic trick

RaumZeitLabor
Mannheim Germany

Ange Albertini

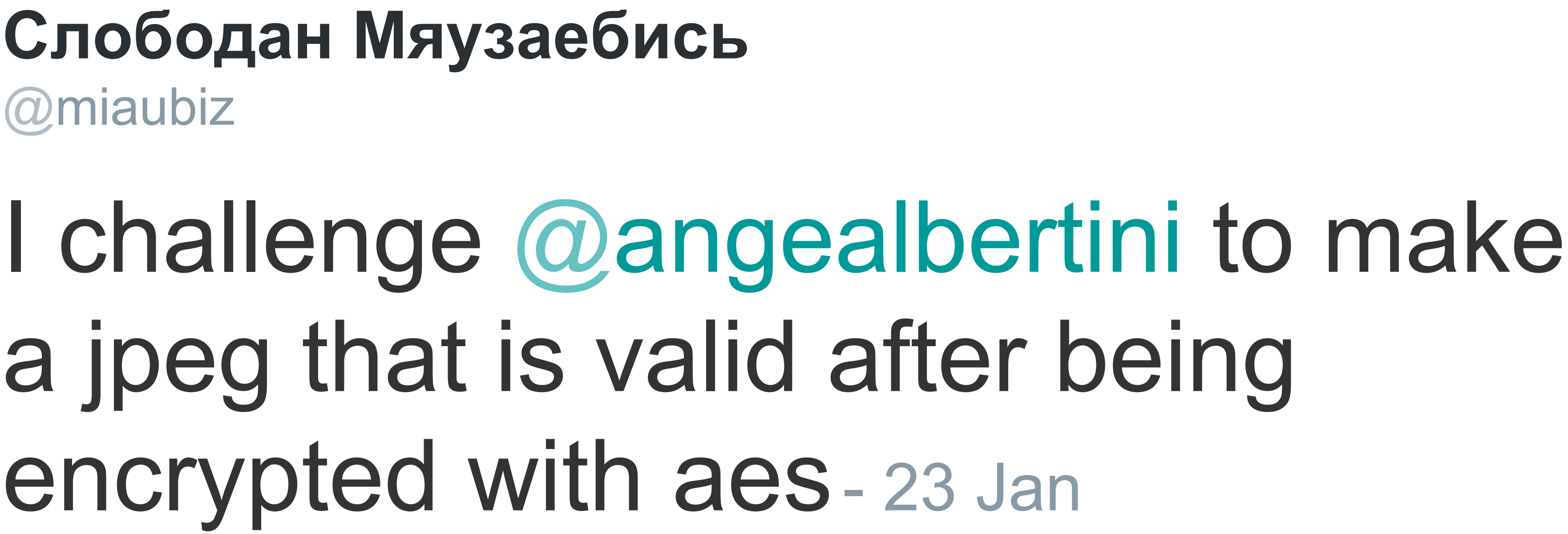
2014/05/17

reverse engineering

&

VISUAL DOCUMENTATIONS

corkami.com



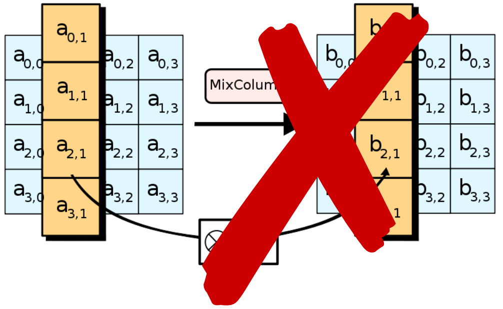
Слободан Мяузаебись
@miaubiz

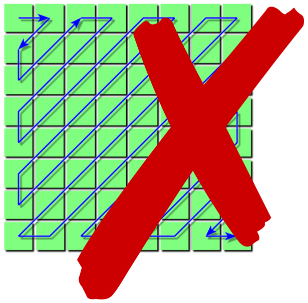
I challenge @angealbertini to make a jpeg that is valid after being encrypted with aes - 23 Jan

the challenge

no need to know AES or JPG

they’re too complex anyway ☺





we’ll just play with lego blocks

let’s keep it simple, and fun

Agenda

- basics
  - crypto basics
  - binary formats basics
- tackle the challenge
- Angecryption
- a walkthrough example
- extra
  - hidden appended data
  - improving ZIP compatibility
  - GynCryption
- conclusion

Crypto basics

block cipher, encryption, plaintext...

AES(*) is a block cipher

like Triple-DES, Blowfish...

(*) from now on we'll say AES for AES-128. it doesn't really matter, just makes the key smaller ☺

A block cipher

- takes a block of data
  - of fixed size (=“block size”)
    - 16 bytes for AES, 8 for Blowfish/DES^3...
  - padded if smaller than blocksize
- a key
- returns a ‘scrambled’ block of data

- security criteria:
  - invertible (permutation)..
  - but only if the key is known
- behaves as a 'random permutation' (aka 'ideal cipher')

AES encryption 1/3

Parameters

k:'MySecretKey12345'

block:'a block of text.'

Results

┐◄n╩i▐☼←∞└╞∙iû╨►

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

AES encryption 2/3

Parameters

k:'MySecretKey12346'

block:'a block of text.'

Results

gO┼╗ÑëΩcë ▼LÇk╨î

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

AES encryption 3/3

Parameters

Results

k:'MySecretKey12345'

block:'a block of text!'

wε╩▬▄y&↕ú@αùαφ♣O

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

with a tiny change in the key or input block, the output block is completely different

we can’t control the output

(the differences are unpredictable)

Reverse operation

- get the original block with the reverse operation and the same key
- encrypt then decrypt

In some ciphers (such as NOEKEON*), encryption and decryption are almost identical.

*http://gro.noekeon.org/

Jargon

plaintext = readable, not encrypted (in theory)

a **plaintext** block is **encrypted** into **ciphertext** block

a **ciphertext** block is **decrypted** into a **plaintext** block

Encryption and decryption 1/3

Encrypting “a block of text.”
with key = “MySecretKey12345”
with AES gives

“┐◄n╩i▐☼←∞└╞·iû╨►” (BF 11 6E CA 69 DE 0F 1B EC C0 C6 F9 69 96 D0 10)

Encryption and decryption 2/3

Decrypting the result (“┐ ◄n╩╚i ▌☼←∞ └╞·iû╨►”) with the same key (“MySecretKey12345”) gives back “a block of text.”

Encryption and decryption 3/3

but decrypting the same block again
with a slightly different key “MySecretKey12346”
gives “π╔6I►♣♫Σ♣╝╤→√çφ╡ ” (E3 C9 36 49 10 05 0E E4 05 BC D1 1A FB 87 ED B5)

we can’t decrypt without the key used to encrypt

file formats basics

signatures, chunks, appended data...

File formats 101

- most files on your system use a standard format.
- some for executables (ran by the OS)
  - very complex - depend on the OS
- some for documents (open by Office, your browser…)
  - “less” complex - depend on the specs only

File formats signatures (& headers)

usually start with a magic signature

- a fixed byte sequence
  - PNG `\x89 PNG\r\n\x1a\n`
  - PDF `%PDF-1.x`
  - FLV `FLV`
  - JPG `\xFF \xD8`
- enforced at offset 0

Why using a magic signature?

- quick identification
- the file is invalid if the signature is missing

Collisions?

- very rare:
  - `0xCAFEBABE`: universal Mach-O **and** JAVA Class
    - recent Mach-O = `0xFEEDFACE` / `0xFEEDFACF`

Typical data structure

formats are made of chunks

- chunks have different names
  - “chunk”, “segment”, “atom”
- structure (**t***ype* **l***ength* **v***alue*)
  1. a type identifier
     - “marker”, “type”, “id”
  2. (typically) their length
  3. the chunk data itself
  4. (sometimes) data’s checksum

Why using a chunk-structure?

- newer chunk types can be ignored for ‘forward compatibility”
- tools can use custom chunks to store extra info while staying standard

Chunks example (simplified)

A valid file:

1. magic signature
2. chunks
    a. header
    b. comment
    c. thumbnail
    d. data
    e. end

some chunks are critical, some aren’t (=ancillary)

Data structure’s end

- like a magic signature, file formats typically have an end marker.
- the end marker is usually a valid chunk with no data, just an ID

Ex, in PNG (using HexII* representation)

```
00 00 00 00  .I .E .N .D ae 42 60 82
```

(length = 0)   IMAGE END   CRC(“IEND”)

* http://corkami.googlecode.com/svn/trunk/src/HexII/

Appended data

most file formats tolerates any data of any length after the end marker

valid file + random data ⇒ still valid

Few formats reject any appended data:

- Java CLASS, Java Archive

A valid binary file

to summarize:

to be valid, a binary file requires:

1. a valid header
   - including a valid magic
2. a valid chunk structure
   - an end chunk

and may be followed by any data if tolerated

Let’s go back to the challenge

(at last)

Encrypt a valid JPG into a valid JPG

(and if possible, any other standard format)

First analysis

since a block cipher’s output is ‘random’, encrypting a valid JPG into a valid JPG seems impossible:

both files can’t even have valid signatures and structures

we would have to control the output of AES (!)

Block cipher modes 101

how block ciphers are applied to files

Encrypting data bigger than a block

how does one apply encryption on a file?

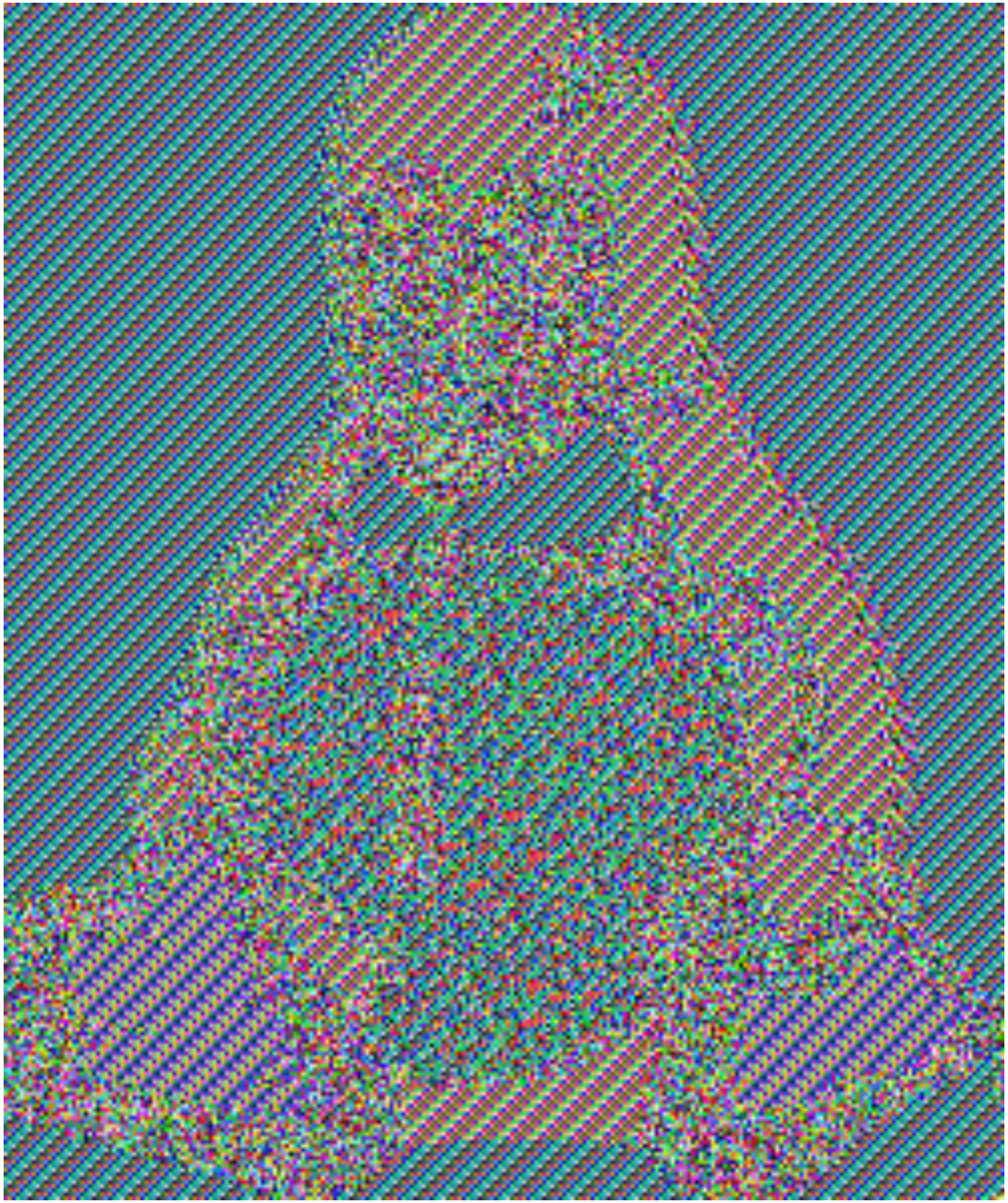
- if the key and plaintext are the same

→ the ciphertext is the same

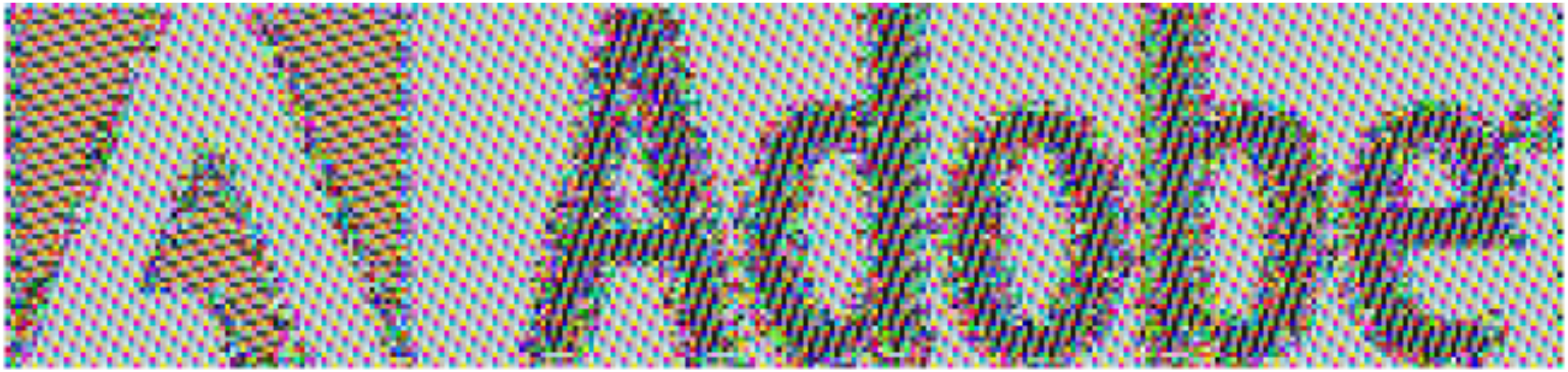
Electronic CodeBook mode

if we just apply the cipher on each block, identical blocks will give identical output

→ big weakness



that doesn’t look terribly encrypted, does it ?



THE ADOBE LOGO, ENCRYPTED WITH 3DES IN ECB MODE
(THE SAME ALGORITHM THEY USE TO STORE PASSWORDS)

Good job, guys!

Block cipher modes of operation

various modes can be used to operate block ciphers on files:

- chaining each block's encryption to propagate differences from the start to the end of the file, killing repetitive patterns

  http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

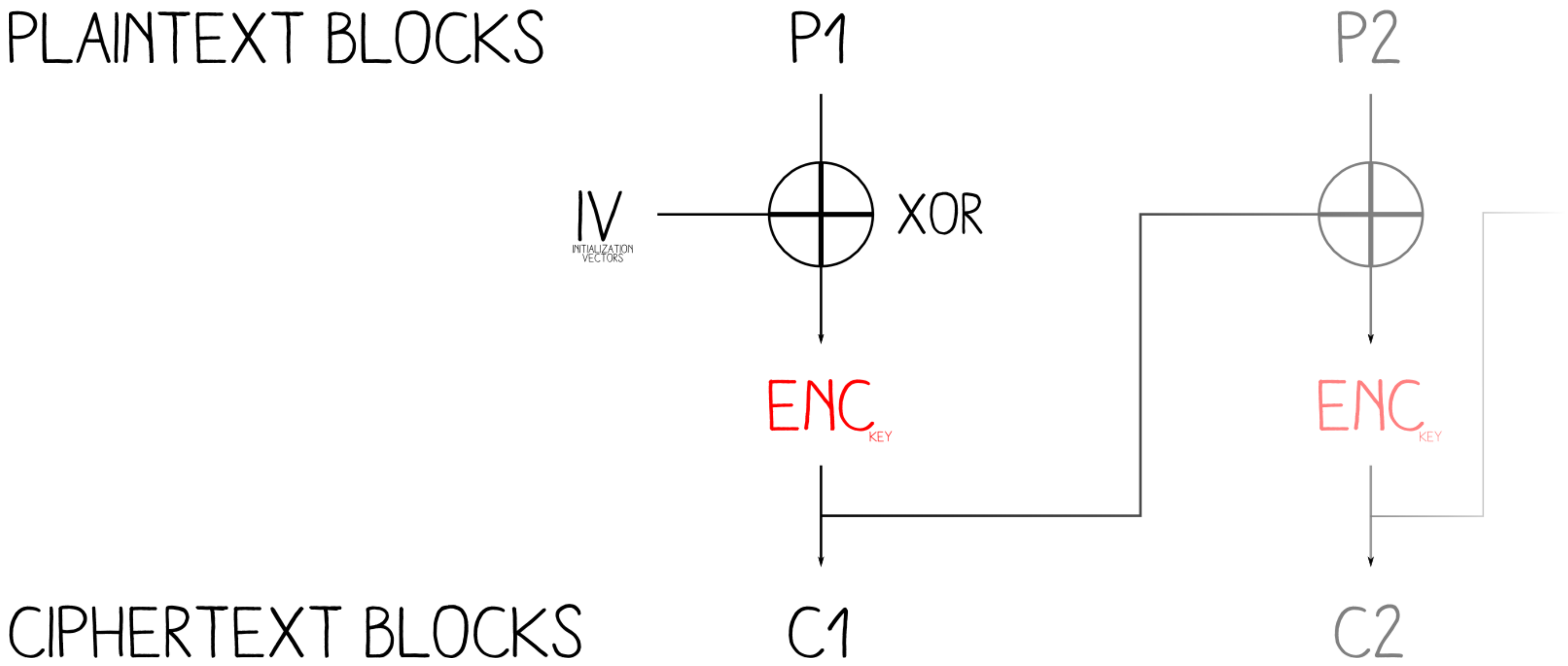
for this, auxiliary input may be needed, such as either:

- unpredictable IV (CBC)
- unique nonce (CTR)

Initialization Vector 101

Several modes (CBC, OFB, CFB,...) introduce an extra parameter *IV* that we can abitrarily choose (in practice, it should be unpredictable)

Cipher Block Chaining



C1 = Enc(P1 ^ IV)

CBC observations

no matter the key or block cipher,

for a given P1 and C1,

we can craft a IV so that:

a file starting with P1 will be encrypted into a file starting with C1

with IV = Dec(C1) xor P1

Example

With key: `my_own_key_12345`

IV: `0f 0d ec 1c 96 4c 5f 1e 84 19 4a 38 81 ef b7 f6`

`"%PDF-1.5\n1 0 obj"`

encrypts as

`"89 PNG 0d 0a 1a 0a 00 00 00 0d IHDR"`

Current status

- we control the first block :)
- the following blocks will look random :(

decrypting plaintext

(ciphers don’t analyze your input)

Encryption & decryption

they are just 2 reverse operations

- they both:
    - take any input
    - give the resulting output
- the reverse operation gives back the original block
    - (if the key is the same)

Example (1/2)

key = "MySecretKey12345"

p = "a block of text."

decrypt(AES, key, p) = “ä/ë-╦7 ↓h│☻⌂µ[←Ñ”

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

it doesn’t really make sense to ‘decrypt’ plaintext…

but it doesn’t matter for the cipher, so...

Example (2/2)

indeed, with:

key = "MySecretKey12345"

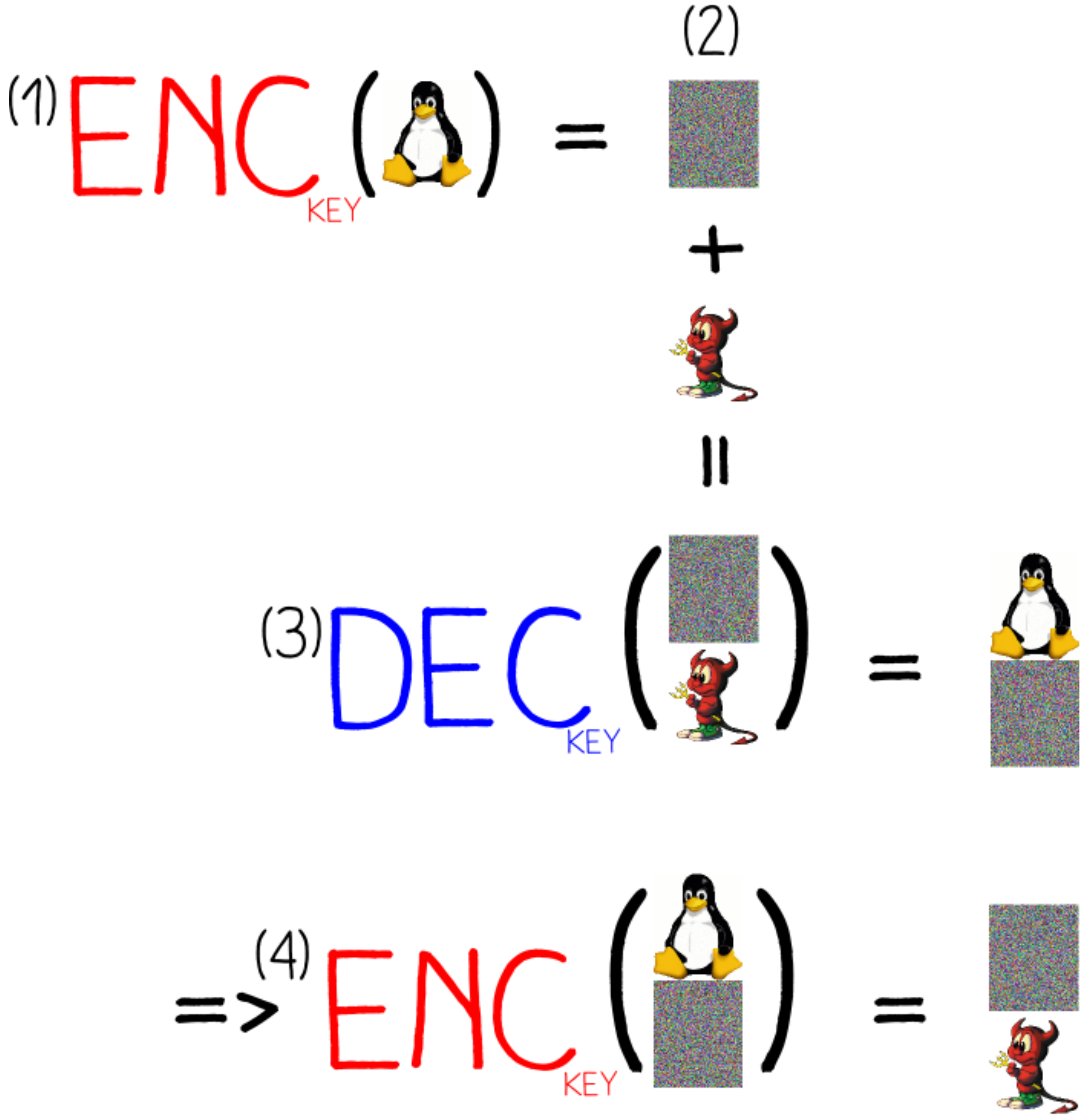
c = “ä/ë-╦7 ↓h│☻⌂µ[←Ñ”

encrypt(AES, key, c) = "a block of text."

you can decrypt plaintext: it gives you back your plaintext after re-encryption

(ie, you can control some AES encryption output)

**let’s add plaintext
to our encrypted file!**


(1)
ENC
KEY
=
(2)
+
=
(3)
DEC
KEY
=
=>
(4)
ENC
KEY
=

Consequences

since adding junk at the end of our valid file
still makes it valid,

we add decrypted plaintext, that will encrypt to what we want

Current status

1. we control the first block
2. we control some appended data

how do we control the encrypted data from the source file that is in-between?

we don’t

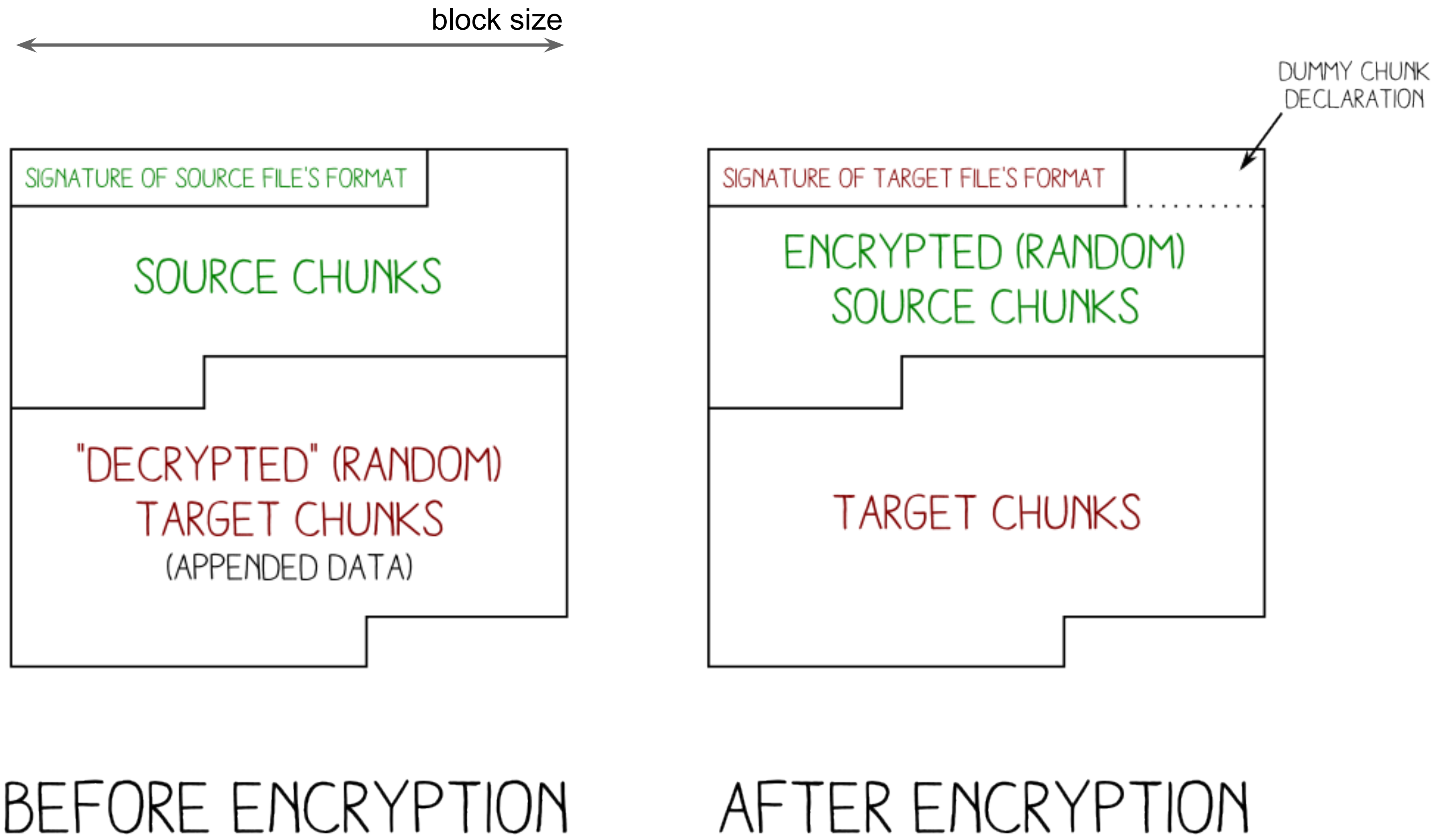
we politely ask the file format to ignore it
(by surrounding this data in an extra chunk)

Our current challenge

within a block, get a valid

1. header
2. chunk start

this is specific to each target format


block size
SIGNATURE OF SOURCE FILE'S FORMAT
SOURCE CHUNKS
"DECRYPTED" (RANDOM)
TARGET CHUNKS
(APPENDED DATA)
DUMMY CHUNK
DECLARATION
SIGNATURE OF TARGET FILE'S FORMAT
ENCRYPTED (RANDOM)
SOURCE CHUNKS
TARGET CHUNKS
BEFORE ENCRYPTION
AFTER ENCRYPTION

our goal

PDF

Portable Document Format

PDF in a nutshell

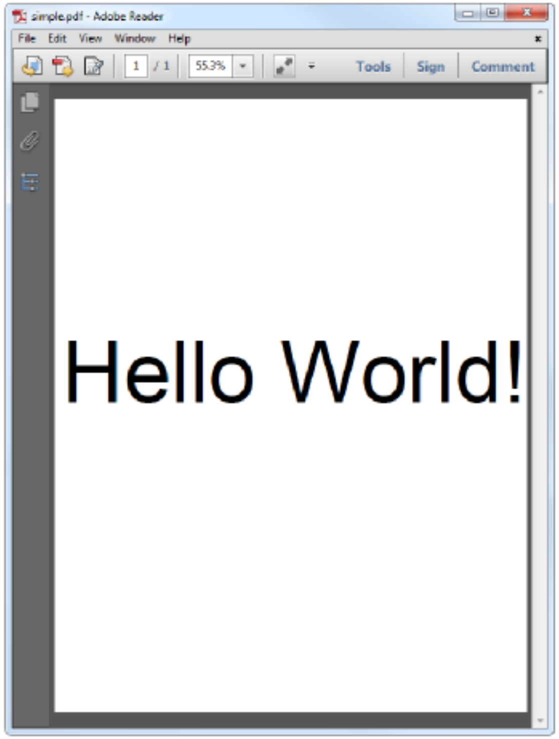
- magic signature: `%PDF-1.X`
- PDF are made of objects
- stream objects can contain any data

```
%PDF-1.1

1 0 obj
<<
  /Pages 2 0 R
>>
endobj

2 0 obj
<<
  /Type /Pages
  /Count 1
  /Kids [3 0 R]
>>
endobj

3 0 obj
<<
  /Type /Page
  /Contents 4 0 R
  /Parent 2 0 R
  /Resources <<
    /Font <<
      /F1 <<
        /Type /Font
        /Subtype /Type1
        /BaseFont /Arial
      >>
    >>
  >>
>>
endobj
```

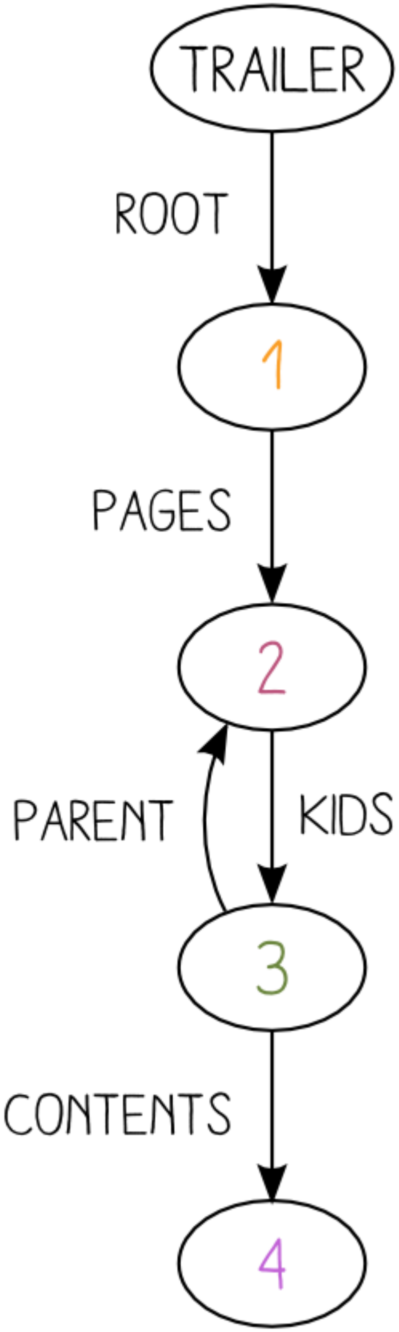
```
4 0 obj
<< /Length 47 >>
stream
BT
  /F1 110
  Tf
  10 400 Td
  (Hello World!)Tj
ET
endstream
endobj
```

```
...
```

```
...

xref
0 5
0000000000 65535 f
0000000010 00000 n
0000000047 00000 n
0000000111 00000 n
0000000313 00000 n

trailer
<<
  /Root 1 0 R
>>

startxref
416
%%EOF
```



Hello World!



Stream objects

<object number> <generation number> **obj**

<< <parameters> **>>**

stream

<data>

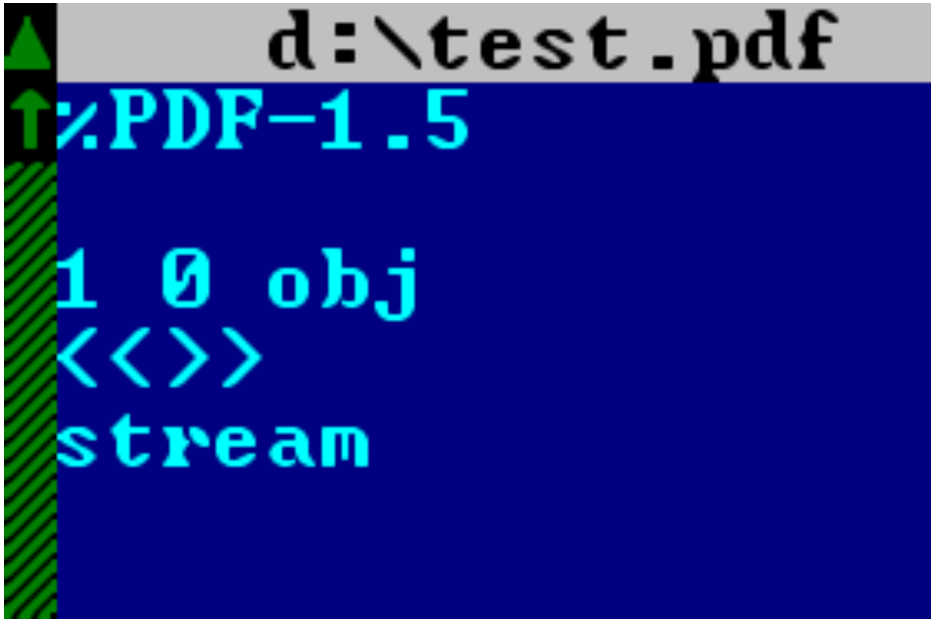
endstream

endobj

Required space for our block

AES has a block size of 16 bytes

a standard PDF header + stream object start takes >30 bytes!



```
%PDF-1.5

1 0 obj
<<>>
stream
```

Let’s shrink the header

1. truncate the signature
   %PDF-\0
2. remove the object number
   ~~0 0 ~~obj
3. remove the parameter dictionary
   ~~<<>>~~

et voilà, **exactly** 16 bytes!

```
%PDF-\0obj\nstream
```

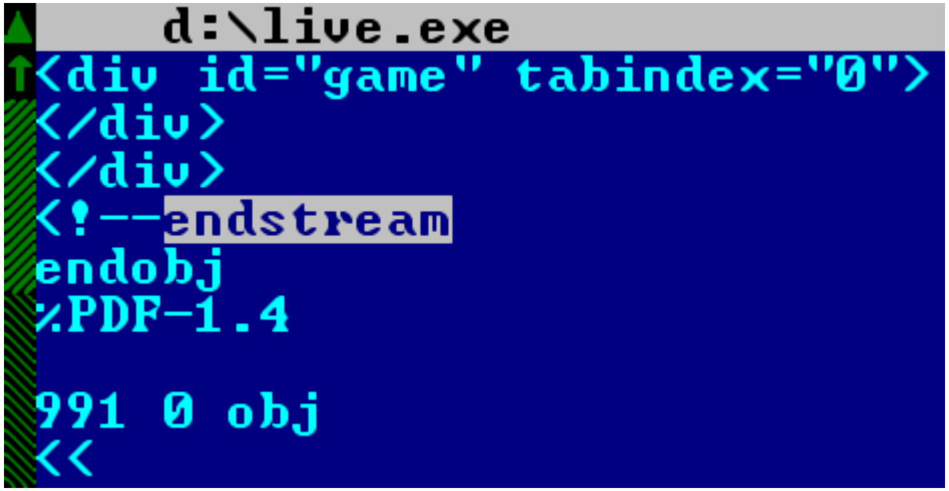
PDF laxism FTW

PDF doesn’t care if 2 signatures are present

→ we can close the stream at *any* point with:

endstream

endobj

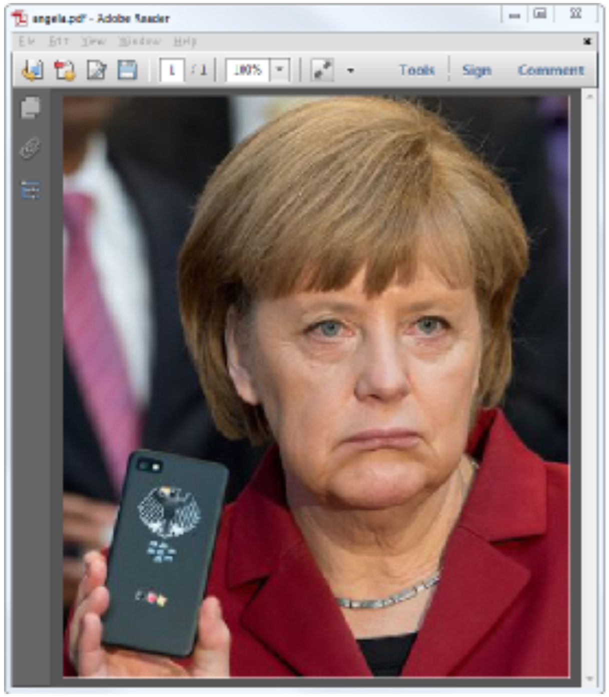


```
<div id="game" tabindex="0">
</div>
</div>
<!--endstream
endobj
%PDF-1.4

991 0 obj
<<
```

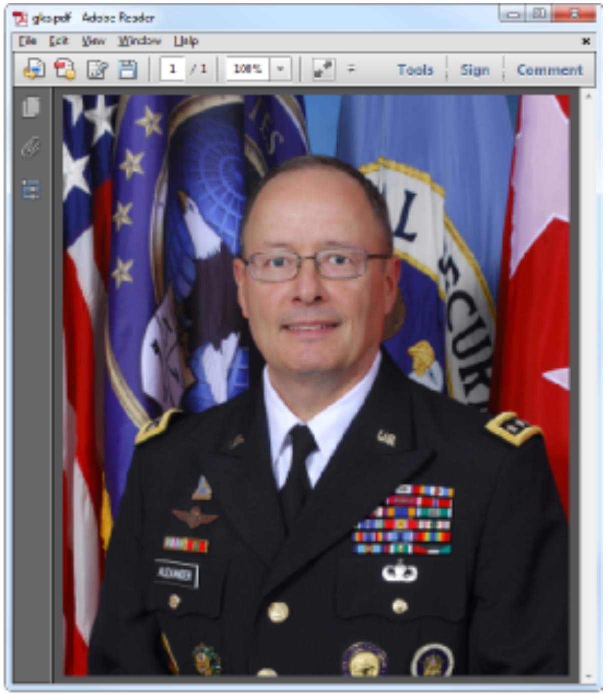
and resume our original PDF file happily

Steps to encrypt as PDF

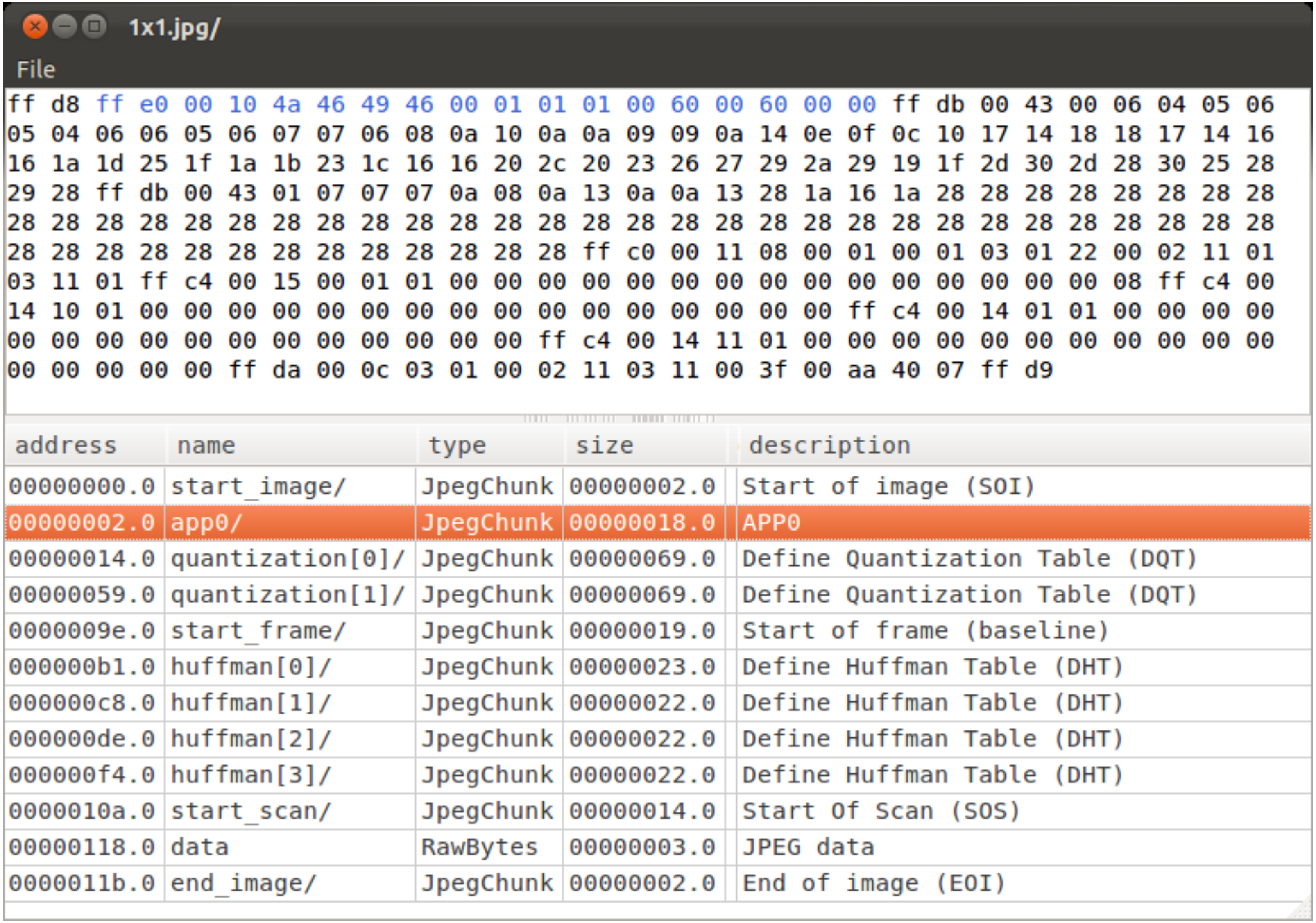
1. we choose our key, source and target contents
2. our first cipher block: `%PDF-\0obj\nstream`
3. determine IV from plaintext & cipher blocks
4. encrypt source file
5. append object termination
6. append target file
7. decrypt final file
8. et voilà, the final file will encrypt as expected!


angela.pdf - Adobe Reader
File Edit View Window Help
1 / 1
100%
Tools
Sign
Comment

=>


gka.pdf - Adobe Reader
File Edit View Window Help
1 / 1
100%
Tools
Sign
Comment

PoC @ corkami



```
ff d8 ff e0 00 10 4a 46 49 46 00 01 01 01 00 60 00 60 00 00 ff db 00 43 00 06 04 05 06
05 04 06 06 05 06 07 07 06 08 0a 10 0a 0a 09 09 0a 14 0e 0f 0c 10 17 14 18 18 17 14 16
16 1a 1d 25 1f 1a 1b 23 1c 16 16 20 2c 20 23 26 27 29 2a 29 19 1f 2d 30 2d 28 30 25 28
29 28 ff db 00 43 01 07 07 07 0a 08 0a 13 0a 0a 13 28 1a 16 1a 28 28 28 28 28 28 28 28
28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28
28 28 28 28 28 28 28 28 28 28 28 28 28 ff c0 00 11 08 00 01 00 01 03 01 22 00 02 11 01
03 11 01 ff c4 00 15 00 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 08 ff c4 00
14 10 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff c4 00 14 01 01 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 ff c4 00 14 11 01 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 ff da 00 0c 03 01 00 02 11 03 11 00 3f 00 aa 40 07 ff d9
```

address	name	type	size	description
00000000.0	start_image/	JpegChunk	00000002.0	Start of image (SOI)
00000002.0	app0/	JpegChunk	00000018.0	APP0
00000014.0	quantization[0]/	JpegChunk	00000069.0	Define Quantization Table (DQT)
00000059.0	quantization[1]/	JpegChunk	00000069.0	Define Quantization Table (DQT)
0000009e.0	start_frame/	JpegChunk	00000019.0	Start of frame (baseline)
000000b1.0	huffman[0]/	JpegChunk	00000023.0	Define Huffman Table (DHT)
000000c8.0	huffman[1]/	JpegChunk	00000022.0	Define Huffman Table (DHT)
000000de.0	huffman[2]/	JpegChunk	00000022.0	Define Huffman Table (DHT)
000000f4.0	huffman[3]/	JpegChunk	00000022.0	Define Huffman Table (DHT)
0000010a.0	start_scan/	JpegChunk	00000014.0	Start Of Scan (SOS)
00000118.0	data	RawBytes	00000003.0	JPEG data
0000011b.0	end_image/	JpegChunk	00000002.0	End of image (EOI)

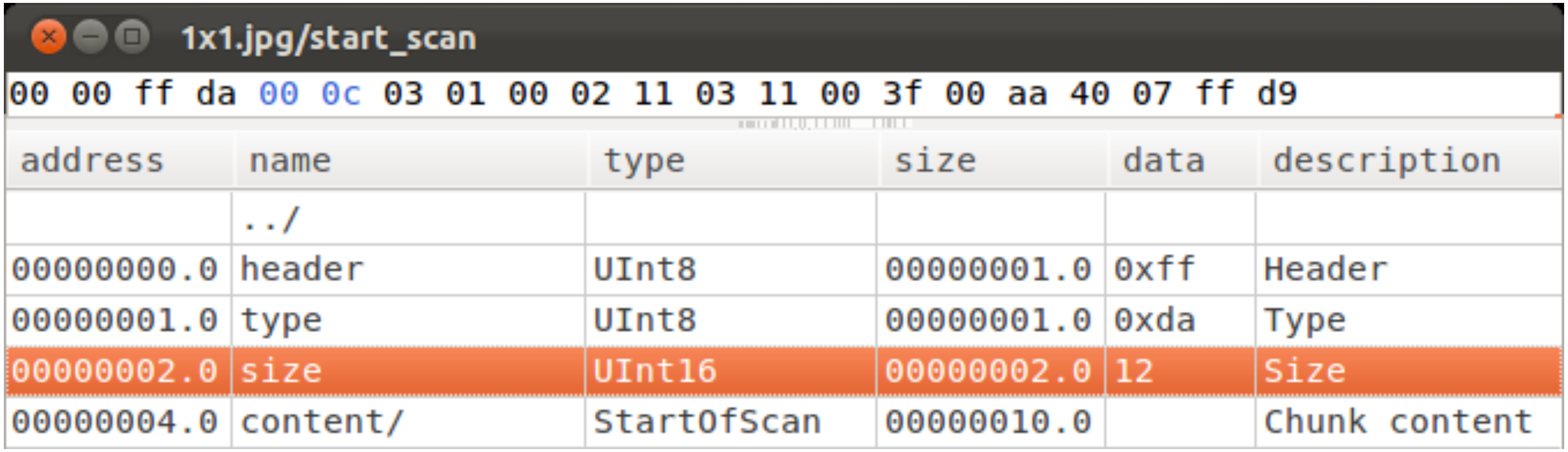
JPG

Joint Photographic Experts Group (image)

JPG in a nutshell

- magic signature: `FF D8` (only 2 bytes)
- chunk’s structure: <id:2> <length:2> <data:?>
- comment chunk ID: `FF FE`

→ only 6 bytes are required!



```
00 00 ff da 00 0c 03 01 00 02 11 03 11 00 3f 00 aa 40 07 ff d9
```

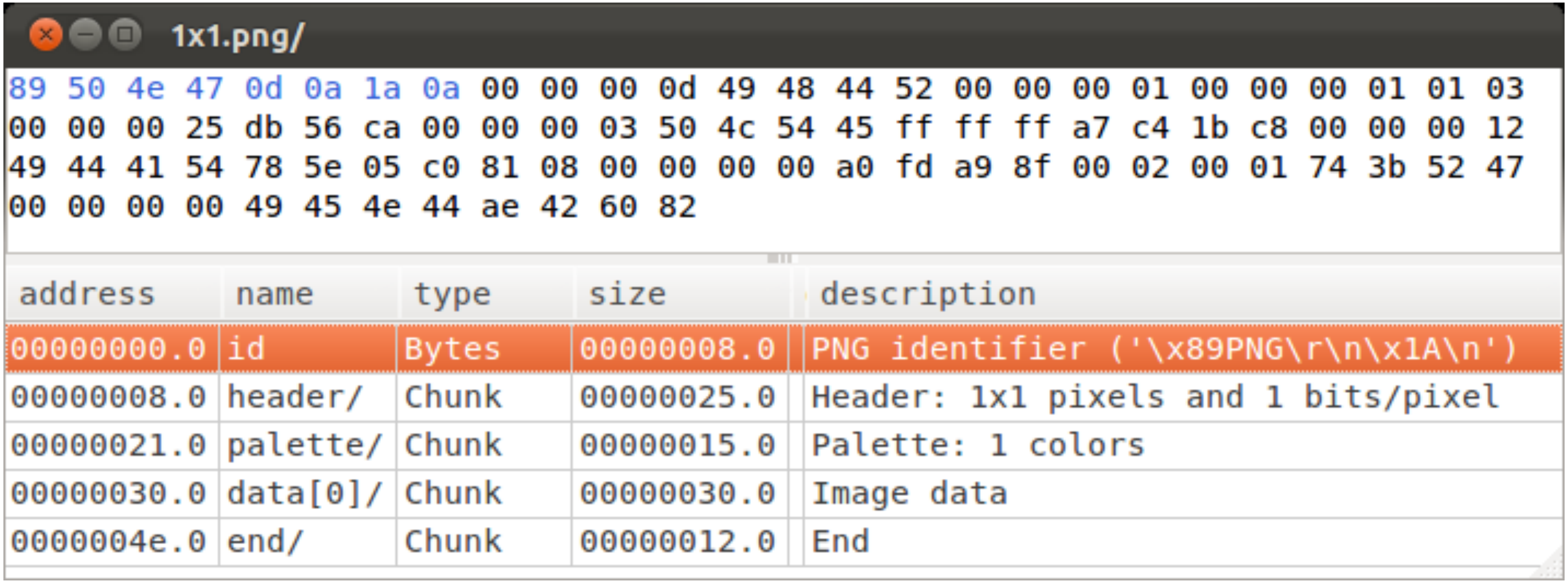
address	name	type	size	data	description
	../				
00000000.0	header	UInt8	00000001.0	0xff	Header
00000001.0	type	UInt8	00000001.0	0xda	Type
00000002.0	size	UInt16	00000002.0	12	Size
00000004.0	content/	StartOfScan	00000010.0		Chunk content

Steps to encrypt as JPG

1. get original size, padded to 16
2. 1st cipher block =

   `FF D8   FF FE` <source size:2> <padding>
3. generate IV from plaintext & cipher blocks
4. AES-CBC encrypt source file
5. append target file minus signature
6. decrypt final file

JPG PoC



```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 00 01 00 00 00 01 01 03
00 00 00 25 db 56 ca 00 00 00 03 50 4c 54 45 ff ff ff a7 c4 1b c8 00 00 00 12
49 44 41 54 78 5e 05 c0 81 08 00 00 00 00 a0 fd a9 8f 00 02 00 01 74 3b 52 47
00 00 00 00 49 45 4e 44 ae 42 60 82
```

address	name	type	size	description
00000000.0	id	Bytes	00000008.0	PNG identifier ('\x89PNG\r\n\x1A\n')
00000008.0	header/	Chunk	00000025.0	Header: 1x1 pixels and 1 bits/pixel
00000021.0	palette/	Chunk	00000015.0	Palette: 1 colors
00000030.0	data[0]/	Chunk	00000030.0	Image data
0000004e.0	end/	Chunk	00000012.0	End

PNG

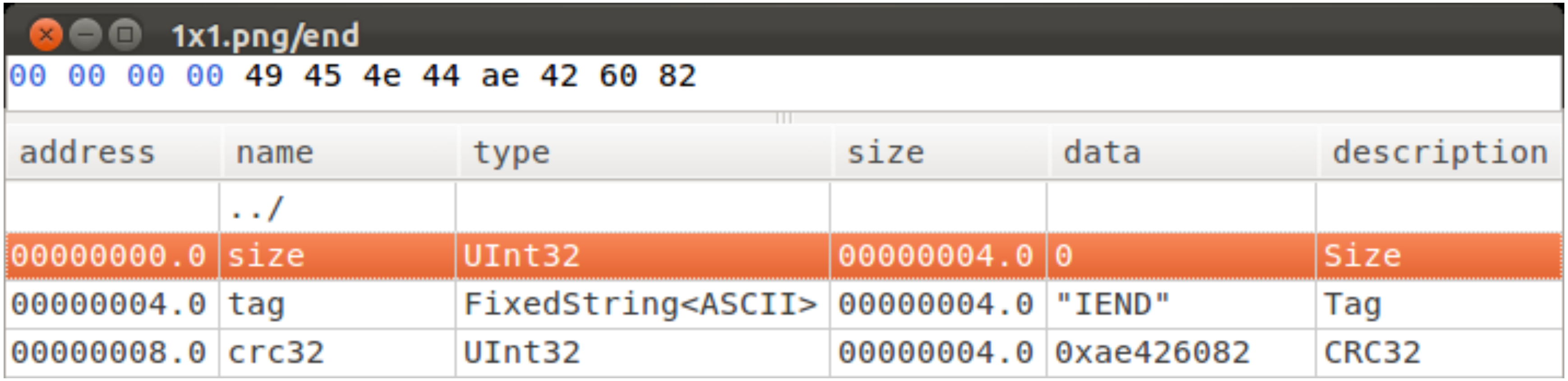
Portable Network Graphics

PNG

- big magic: `\x89PNG\r\n\x1a\n` (8 bytes!)
- chunk's structure:

<length(data):4> <id:4> <data:?> <crc(data+id):4>

signature + chunk declaration = 16 bytes (!)



```
00 00 00 00 49 45 4e 44 ae 42 60 82
```

address	name	type	size	data	description
	../				
00000000.0	size	UInt32	00000004.0	0	Size
00000004.0	tag	FixedString<ASCII>	00000004.0	"IEND"	Tag
00000008.0	crc32	UInt32	00000004.0	0xae426082	CRC32

Encrypt as PNG

1. get original file size
2. generate cipher block
3. compute the IV
4. encrypt original data
5. get encrypted(original data) checksum
6. append checksum and target data
   - target data = target file - signature
7. decrypt file

(1)

PNG SIGNATURE

```
89 .P .N .G 0d 0a 1a 0a
```

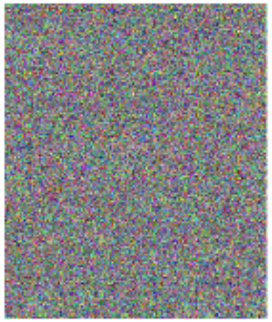
STARTING A DUMMY CHUNK

```
.. .. .. .. .. .. .. .. xx xx xx xx tt tt tt tt
```

CHUNK LENGTH

CHUNK TYPE

RANDOM ENCRYPTED DATA



ENDING DUMMY CHUNK

```
yy yy yy yy
```

CHUNK CRC

(2)

STARTING CONTROLLED DATA

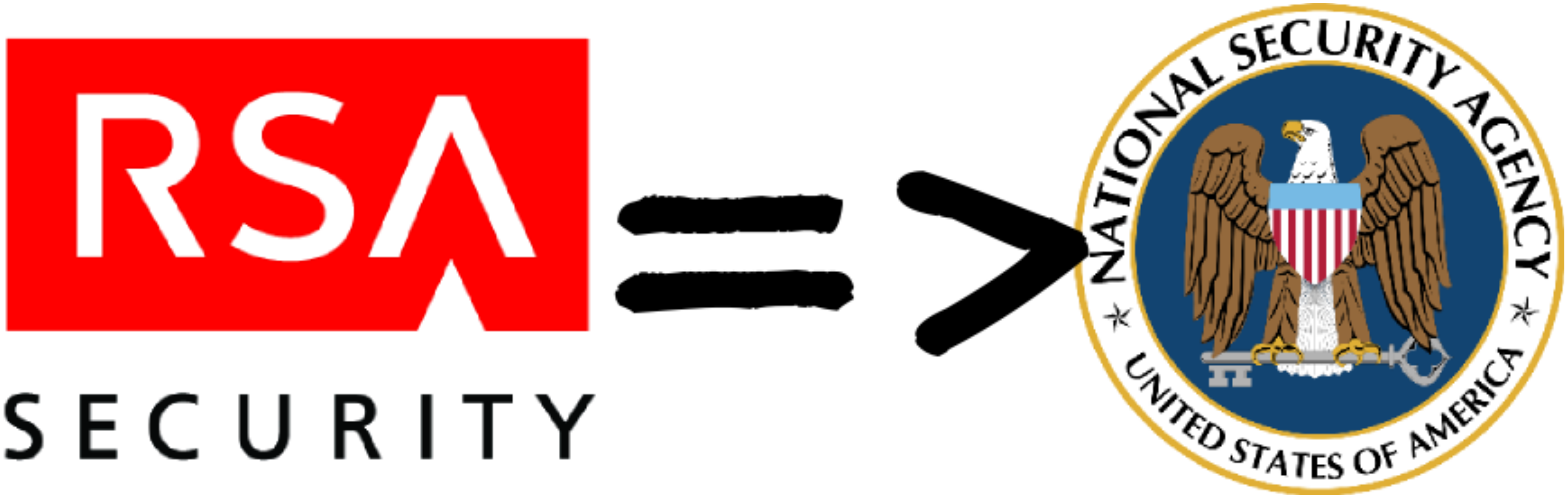
```
.. .. .. .. 00 00 00 0d .I .H .D .R
```

ORIGINAL IMAGE HEADER

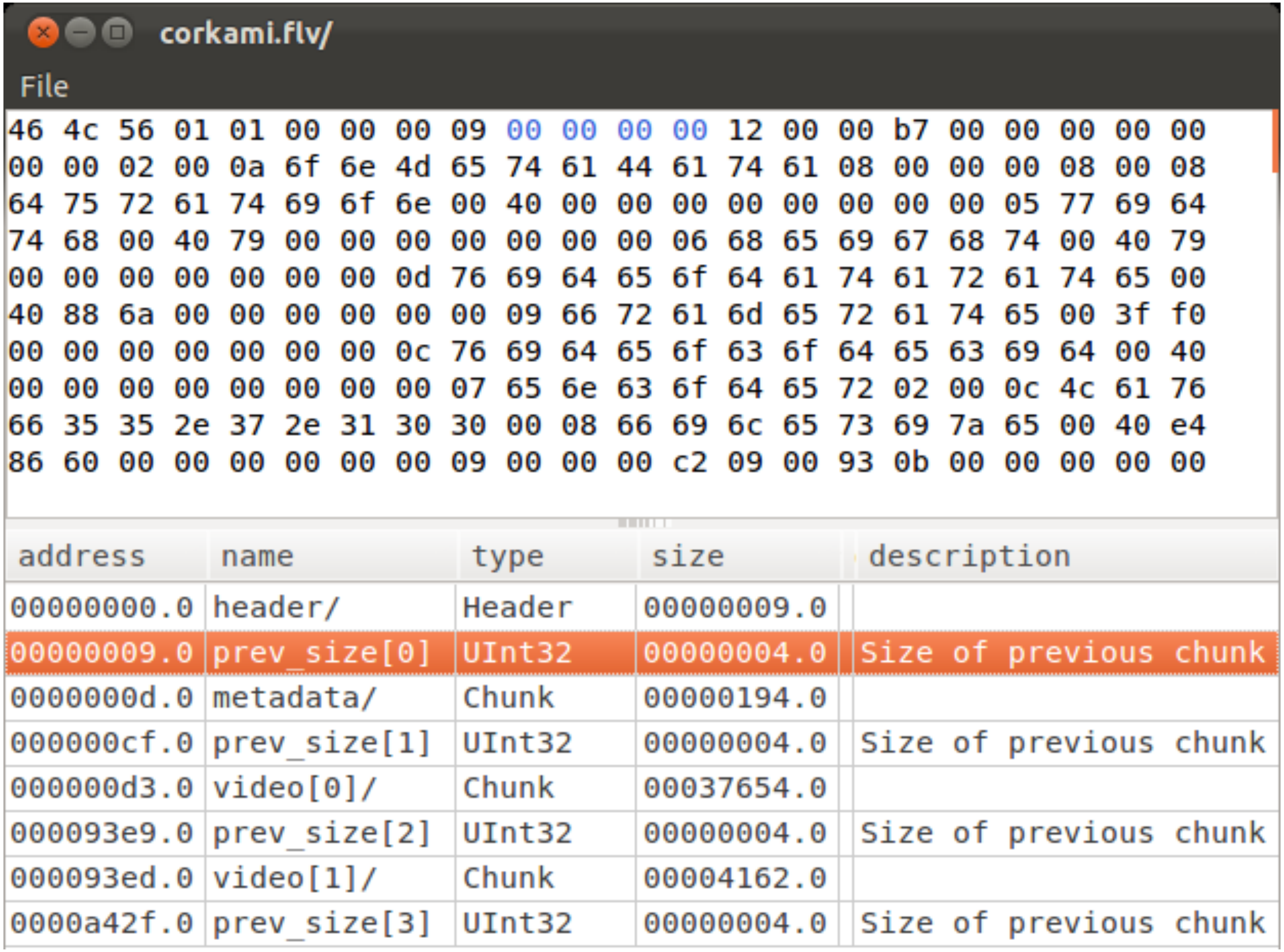
END OF IMAGE

```
...00 00 00 00 .I .E .N .D AE 42 60 82
```

PoC


RSA
SECURITY
NATIONAL SECURITY AGENCY
UNITED STATES OF AMERICA

PNG PoC



```
46 4c 56 01 01 00 00 00 09 00 00 00 00 12 00 00 b7 00 00 00 00 00
00 00 02 00 0a 6f 6e 4d 65 74 61 44 61 74 61 08 00 00 00 08 00 08
64 75 72 61 74 69 6f 6e 00 40 00 00 00 00 00 00 00 00 05 77 69 64
74 68 00 40 79 00 00 00 00 00 00 00 06 68 65 69 67 68 74 00 40 79
00 00 00 00 00 00 00 0d 76 69 64 65 6f 64 61 74 61 72 61 74 65 00
40 88 6a 00 00 00 00 00 00 09 66 72 61 6d 65 72 61 74 65 00 3f f0
00 00 00 00 00 00 00 0c 76 69 64 65 6f 63 6f 64 65 63 69 64 00 40
00 00 00 00 00 00 00 00 07 65 6e 63 6f 64 65 72 02 00 0c 4c 61 76
66 35 35 2e 37 2e 31 30 30 00 08 66 69 6c 65 73 69 7a 65 00 40 e4
86 60 00 00 00 00 00 00 09 00 00 00 c2 09 00 93 0b 00 00 00 00 00
```

address	name	type	size	description
00000000.0	header/	Header	00000009.0	
00000009.0	prev_size[0]	UInt32	00000004.0	Size of previous chunk
0000000d.0	metadata/	Chunk	00000194.0	
000000cf.0	prev_size[1]	UInt32	00000004.0	Size of previous chunk
000000d3.0	video[0]/	Chunk	00037654.0	
000093e9.0	prev_size[2]	UInt32	00000004.0	Size of previous chunk
000093ed.0	video[1]/	Chunk	00004162.0	
0000a42f.0	prev_size[3]	UInt32	00000004.0	Size of previous chunk

FLV

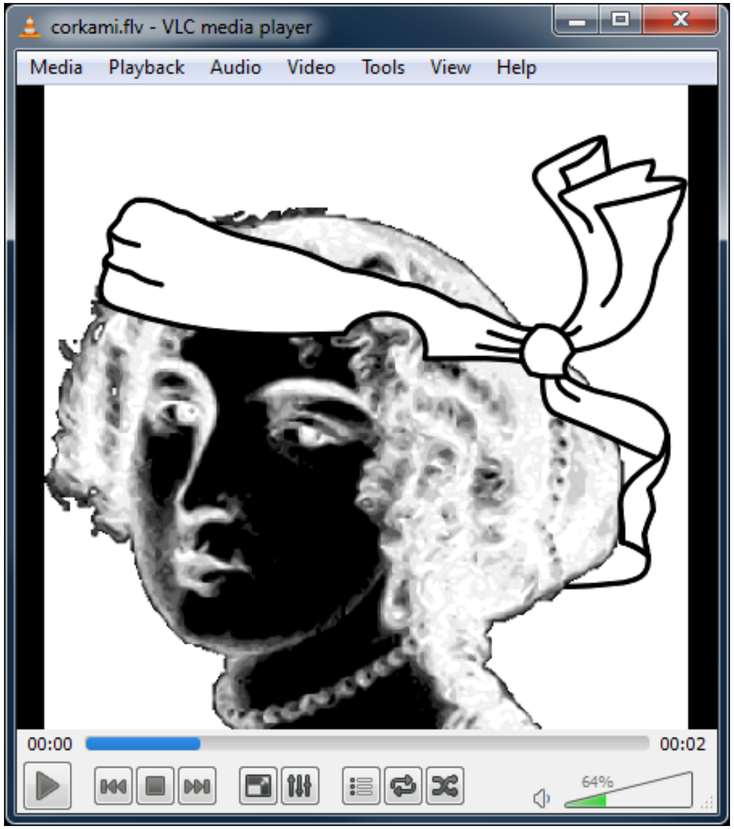
Flash Video

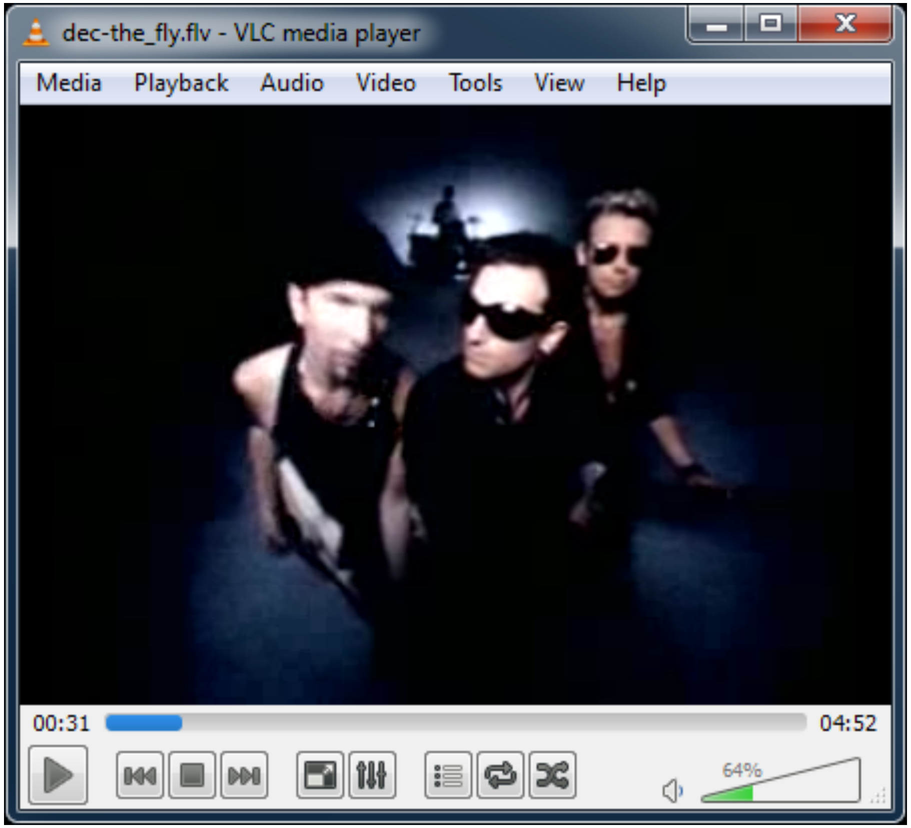
Flash Video

1. magic = “FLV”
2. followed by 2 bytes parameters
3. then **size(chunk)** on 4 bytes

⇒ we can arbitrarily increase it
and put our next chunk where we want

no checksum or trick

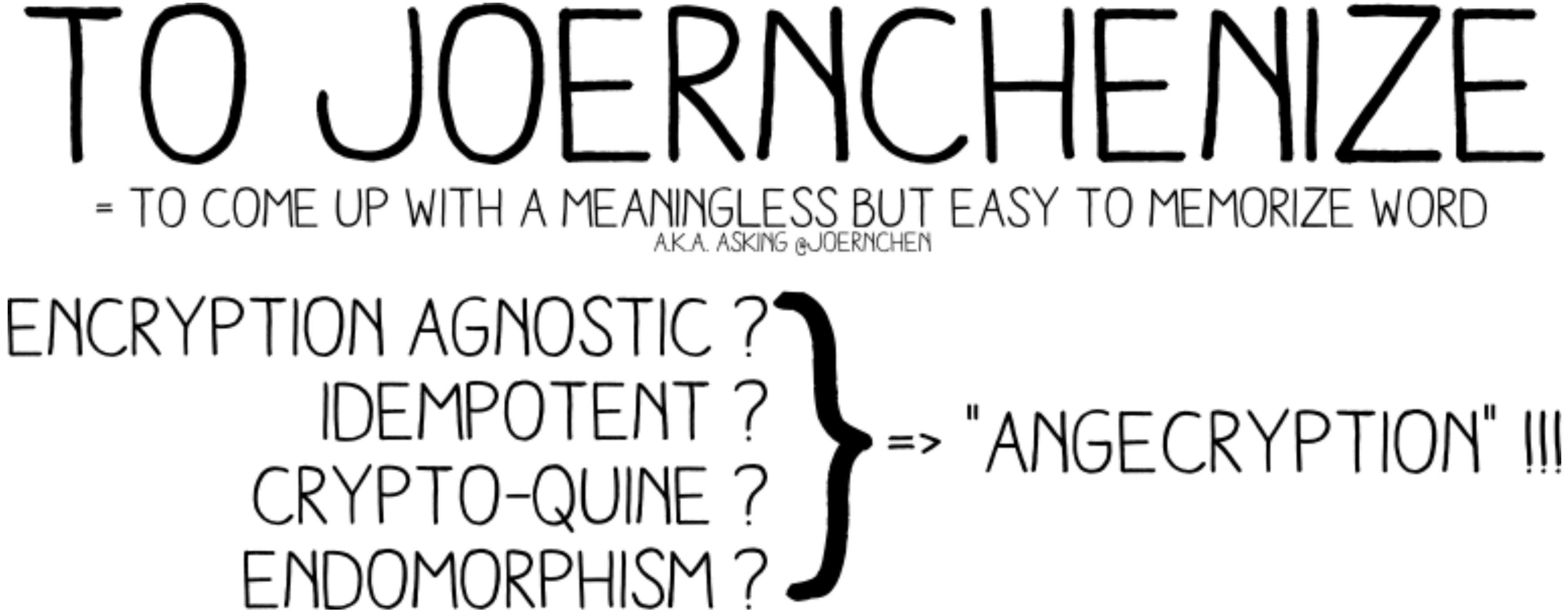

corkami.flv - VLC media player
Media Playback Audio Video Tools View Help
00:00
00:02
64%


dec-the_fly.flv - VLC media player
Media Playback Audio Video Tools View Help
00:31
04:52
64%

an FLV PoC

(key = “a man will crawl”)

How can we call that trick?



TO JOERNCHENIZE

= TO COME UP WITH A MEANINGLESS BUT EASY TO MEMORIZE WORD

A.K.A. ASKING @JOERNCHEN

ENCRYPTION AGNOSTIC ?
IDEMPOTENT ?
CRYPTO-QUINE ?
ENDOMORPHISM ?

=> "ANGECRYPTION" !!!

Reminder

- this is not specific to AES
- this is not specific to CBC

required conditions

- control the first cipherblock
- the source format tolerates appended data
- header+chunk declaration fits in “blocksize”
  - the source size fits in the specified size encoding (short, long…)

Bonus

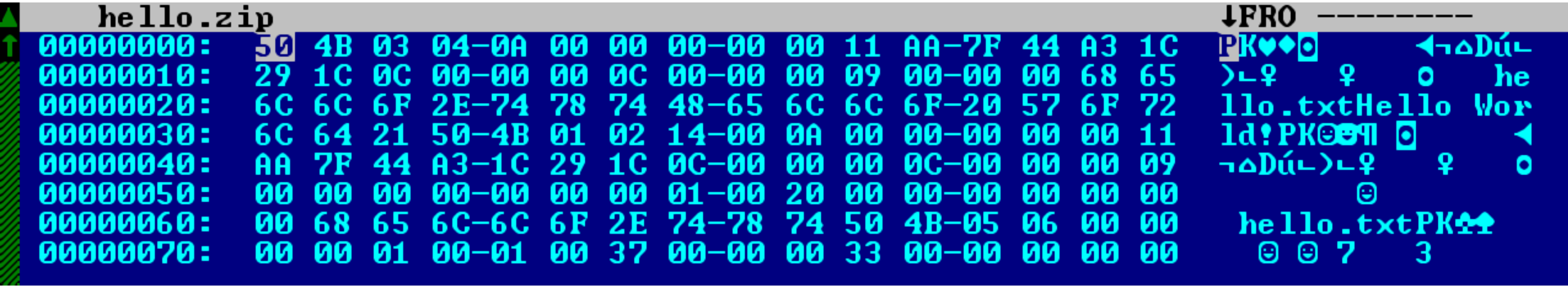
as a consequence

- the same file can encrypt or decrypt to
    - various files
    - of different formats
    - with different ciphers
    - and different modes if you can craft a header (see GynCryption)

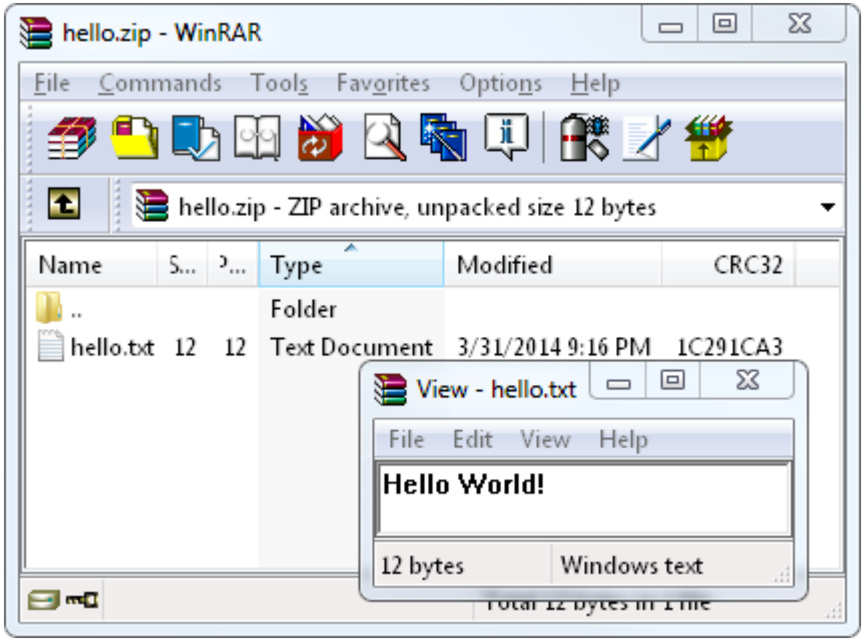
a step by step walkthrough

AES(ZIP) = PNG

Let’s encrypt this (ZIP)



```
     hello.zip                                                                    ↓FRO ---------
00000000:   50 4B 03 04-0A 00 00 00-00 00 11 AA-7F 44 A3 1C   PK♥♦◙     ◄¬⌂Dú∟
00000010:   29 1C 0C 00-00 00 0C 00-00 00 09 00-00 00 68 65   )∟♀   ♀   ○   he
00000020:   6C 6C 6F 2E-74 78 74 48-65 6C 6C 6F-20 57 6F 72   llo.txtHello Wor
00000030:   6C 64 21 50-4B 01 02 14-00 0A 00 00-00 00 00 11   ld!PK☺☻¶ ◙     ◄
00000040:   AA 7F 44 A3-1C 29 1C 0C-00 00 00 0C-00 00 00 09   ¬⌂Dú∟)∟♀   ♀   ○
00000050:   00 00 00 00-00 00 01 00-20 00 00 00-00 00 00 00         ☺
00000060:   00 68 65 6C-6C 6F 2E 74-78 74 50 4B-05 06 00 00    hello.txtPK♣♠
00000070:   00 00 01 00-01 00 37 00-00 00 33 00-00 00 00 00     ☺ ☺ 7   3
```



hello.zip - WinRAR

hello.zip - ZIP archive, unpacked size 12 bytes

Name	S...	P...	Type	Modified	CRC32
..			Folder		
hello.txt	12	12	Text Document	3/31/2014 9:16 PM	1C291CA3

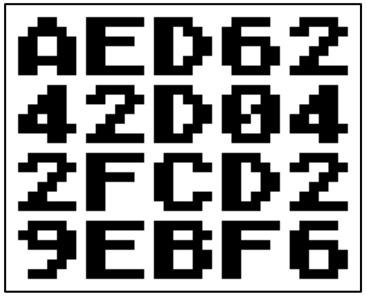
View - hello.txt

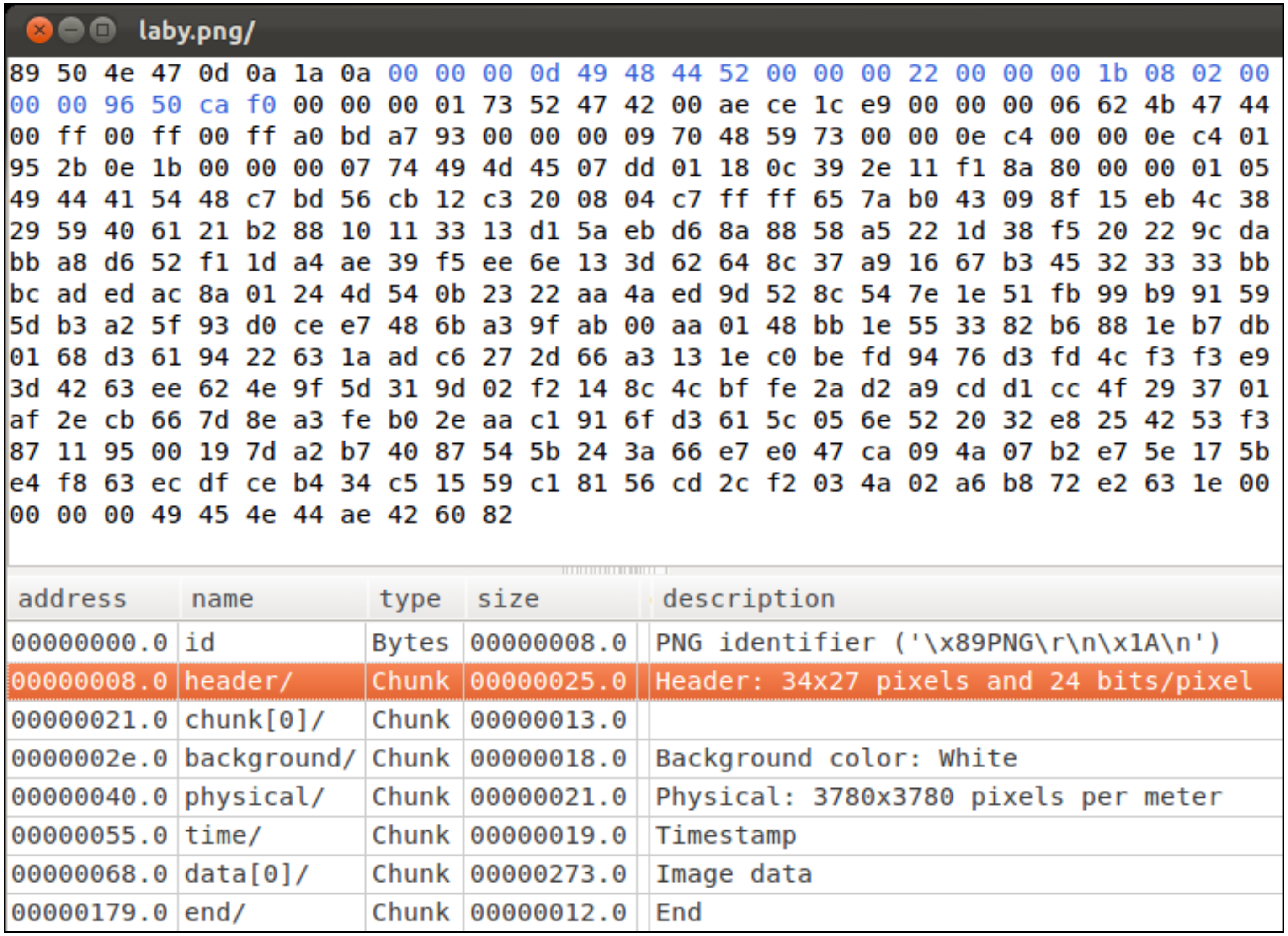
Hello World!

12 bytes | Windows text

Total 12 bytes in 1 file

Into this (PNG)





```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 00 22 00 00 00 1b 08 02 00
00 00 96 50 ca f0 00 00 00 01 73 52 47 42 00 ae ce 1c e9 00 00 00 06 62 4b 47 44
00 ff 00 ff 00 ff a0 bd a7 93 00 00 00 09 70 48 59 73 00 00 0e c4 00 00 0e c4 01
95 2b 0e 1b 00 00 00 07 74 49 4d 45 07 dd 01 18 0c 39 2e 11 f1 8a 80 00 00 01 05
49 44 41 54 48 c7 bd 56 cb 12 c3 20 08 04 c7 ff ff 65 7a b0 43 09 8f 15 eb 4c 38
29 59 40 61 21 b2 88 10 11 33 13 d1 5a eb d6 8a 88 58 a5 22 1d 38 f5 20 22 9c da
bb a8 d6 52 f1 1d a4 ae 39 f5 ee 6e 13 3d 62 64 8c 37 a9 16 67 b3 45 32 33 33 bb
bc ad ed ac 8a 01 24 4d 54 0b 23 22 aa 4a ed 9d 52 8c 54 7e 1e 51 fb 99 b9 91 59
5d b3 a2 5f 93 d0 ce e7 48 6b a3 9f ab 00 aa 01 48 bb 1e 55 33 82 b6 88 1e b7 db
01 68 d3 61 94 22 63 1a ad c6 27 2d 66 a3 13 1e c0 be fd 94 76 d3 fd 4c f3 f3 e9
3d 42 63 ee 62 4e 9f 5d 31 9d 02 f2 14 8c 4c bf fe 2a d2 a9 cd d1 cc 4f 29 37 01
af 2e cb 66 7d 8e a3 fe b0 2e aa c1 91 6f d3 61 5c 05 6e 52 20 32 e8 25 42 53 f3
87 11 95 00 19 7d a2 b7 40 87 54 5b 24 3a 66 e7 e0 47 ca 09 4a 07 b2 e7 5e 17 5b
e4 f8 63 ec df ce b4 34 c5 15 59 c1 81 56 cd 2c f2 03 4a 02 a6 b8 72 e2 63 1e 00
00 00 00 49 45 4e 44 ae 42 60 82
```

address	name	type	size	description
00000000.0	id	Bytes	00000008.0	PNG identifier ('\x89PNG\r\n\x1A\n')
00000008.0	header/	Chunk	00000025.0	Header: 34x27 pixels and 24 bits/pixel
00000021.0	chunk[0]/	Chunk	00000013.0	
0000002e.0	background/	Chunk	00000018.0	Background color: White
00000040.0	physical/	Chunk	00000021.0	Physical: 3780x3780 pixels per meter
00000055.0	time/	Chunk	00000019.0	Timestamp
00000068.0	data[0]/	Chunk	00000273.0	Image data
00000179.0	end/	Chunk	00000012.0	End

Preliminary

- ZIP tolerates appended data, so does PNG

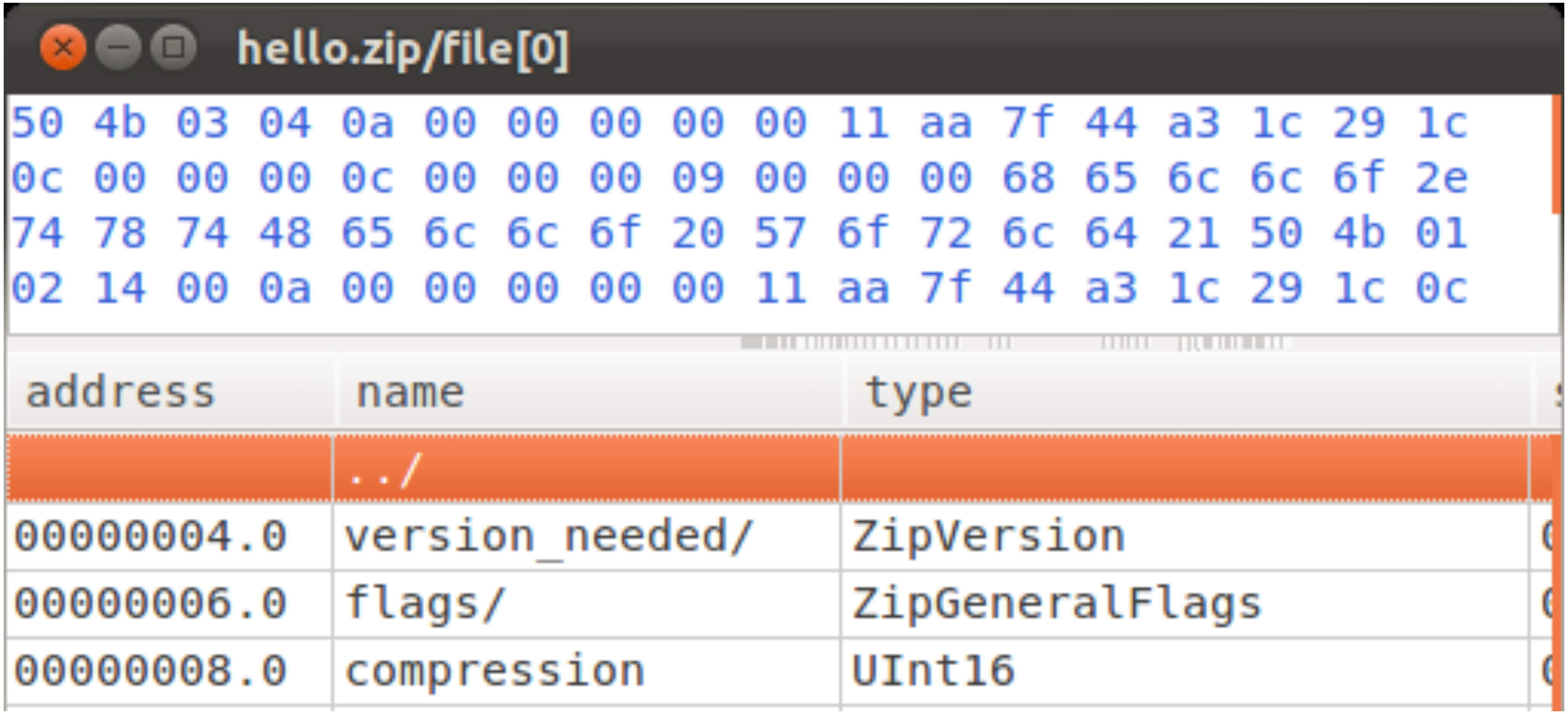
- our source file is 128 bytes
- AES works with 16 bytes blocks

→ one block of 16 bytes of value 0x10 will be padded (not strictly required here, but that's the standard PKCS7 padding)

P1

the first block of the source file is:

```
.P .K 03 04 0A 00 00 00 00 00 11 AA 7F 44 A3 1C
```



```
50 4b 03 04 0a 00 00 00 00 00 11 aa 7f 44 a3 1c 29 1c
0c 00 00 00 0c 00 00 00 09 00 00 00 68 65 6c 6c 6f 2e
74 78 74 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 50 4b 01
02 14 00 0a 00 00 00 00 00 11 aa 7f 44 a3 1c 29 1c 0c
```

address	name	type
	../	
00000004.0	version_needed/	ZipVersion
00000006.0	flags/	ZipGeneralFlags
00000008.0	compression	UInt16

Target format 1/2

the target format is a PNG:

- the encrypted file must start with the PNG signature:

  `89 .P .N .G \r \n 1A \n` (8 bytes)
- followed by chunk length
  - our source file is 144 bytes (with padding)
  - already 16 bytes are covered by first block
  - so our dummy block will be 128 bytes long
  - encoded `00 00 00 80`, as PNG is little endian

Target format 2/2

- followed by chunk type
  - 4 letters, non-critical if starting with lowercase
    - we could use the standard ‘`tEXt`’ comment chunk
    - or just our own, ‘`aaaa`’ or whatever

so our target’s first cipherblock will be:

```
89 .P .N .G \r \n 1A \n   00 00 00 80   61 61 61 61
SIG --------------------   LENGTH ----   TYPE ------
```

Decrypting C1

- the key we’ll use is: `MySecretKey01234`
- our C1 is:

```
89 .P .N .G \r \n 1A \n 00 00 00 80 61 61 61 61
```

- with this key, C1 decrypts as:

```
ee 1b 01 b2 5a a5 bd a8 3a 9e 35 44 2f 5f 23 35
```

Crafting the IV

- P1 is:

```
.P .K 03 04 0A 00 00 00 00 00 11 AA 7F 44 A3 1C
```

- our decrypted C1 is:

```
89 .P .N .G \r \n 1A \n 00 00 00 80 61 61 61 61
```

- by xoring them, we get the IV:

```
be 50 02 b6 50 a5 bd a8 3a 9e 24 ee 50 1b 80 29
```

now, our key and IV are determined.
we just need to combine both file’s content.

Making the final file

1. encrypt our padded source file
2. determine the CRC of our dummy chunk once encrypted (even if it will be surrounded by ‘plaintext’):
    - `6487910E` in our case
3. append this CRC to finish the chunk
4. append all the chunks (whole file minus the SIG) of the target file.

→ our file is now a valid PNG

Our file

1. original source file
2. padding
3. ‘decrypted’ target content

= source file + appended data

```
50 4B 03 04-0A 00 00 00-00 00 11 AA-7F 44 A3 1C  PK???     ?¬¦Dú?
29 1C 0C 00-00 00 0C 00-00 00 09 00-00 00 68 65  )??   ?   ?   he
6C 6C 6F 2E-74 78 74 48-65 6C 6C 6F-20 57 6F 72  llo.txtHello Wor
6C 64 21 50-4B 01 02 14-00 0A 00 00-00 00 00 11  ld!PK??¶ ?     ?
AA 7F 44 A3-1C 29 1C 0C-00 00 00 0C-00 00 00 09  ¬¦Dú?)??   ?   ?
00 00 00 00-00 00 00 01-00 20 00 00-00 00 00 00          ?
00 68 65 6C-6C 6F 2E 74-78 74 50 4B-05 06 00 00   hello.txtPK??
00 00 01 00-01 00 37 00-00 00 33 00-00 00 00 00    ? ? 7   3
10 10 10 10-10 10 10 10-10 10 10 10-10 10 10 10  ????????????????
AA 81 13 6A-22 E8 E3 13-E8 BB 56 83-4D 6D 6A E5  ¬ü?j"Fp?F+VâMmjs
96 DE 62 C6-21 11 52 51-60 C4 E4 19-0E 6E 7F FC  û¦b¦!?RQ`-S??n¦n
F0 37 F6 33-AD E0 42 49-21 B5 1C FB-50 EE E1 6D  =7÷3¡aBI!¦?vPeßm
D3 4F 22 43-DB A9 18 2D-0F EC B5 52-F3 A4 8C EE  +O"C¦¬?-¤8¦R=ñîe
69 A8 E4 5A-96 46 4A 3B-5D E2 B6 8F-4E A6 E7 90  i¿SZûFJ;]G¦ÅNªtÉ
CA E9 E1 04-65 24 D3 49-55 DF AC 68-A1 FC 0F 0F  -Tß?e$+IU¯¼hín¤¤
63 7A 2B A4-26 99 13 22-8A 8B 14 08-8D 71 18 83  cz+ñ&Ö?"èï¶?ìq?â
00 A9 85 86-A6 EC 13 9F-9E 16 30 1A-58 56 B5 CC   ¬àåª8?ƒP?0?XV¦¦
73 77 42 99-EC 53 D8 7C-8C 13 3E 74-6F B2 66 1D  swBÖ8S+|î?>to¦f?
7E CA 62 94-6D B2 D7 E4-F0 21 F5 87-AA F3 F7 8C  ~-böm¦+S=!)ç¬=˜î
15 B9 8D F0-DF FA 56 A3-06 A1 07 25-D1 DC 9D 51  §¦ì=¯·Vú?í•%-_¥Q
F4 6C 7B 43-40 32 57 C8-FD 40 A0 98-CA 6E 02 2B  (l{C@2W+²@áÿ-n?+
6D 54 37 7C-0A 1A C5 DD-9D CC C1 8A-72 A7 FD 24  mT7|??+¦¥¦-èr°²$
12 5F 51 84-4B 48 C3 5D-E0 76 8B 05-8F 09 20 17  ?_QäKH+]avï?Å? ?
A5 BD CE DF-E8 B3 E8 5B-CD 76 63 29-C0 77 BF 28  Ñ++¯F¦F[-vc)+w+(
96 FD 32 05-F8 B6 A3 A9-24 2C A6 98-71 6A 83 DC  û²2?°¦ú¬$,ªÿqjâ_
FE 54 EA ED-43 12 12 EF-BB 38 6E 17-59 17 AF 17  ¦TOfC??n+8n?Y?»?
A9 0C 25 F2-19 11 2C 45-5E 40 77 33-10 09 CE BD  ¬?%=??,E^@w3??++
61 CE 65 BB-8E E6 EE 3E-D5 78 29 85-1D F8 3A 39  a+e+Äµe>+x)à?°:9
85 B0 37 79-01 AF 7F 79-D8 60 1B 59-54 8D A6 03  à¦7y?»¦y+`?YTìª?
93 B9 DF 53-83 47 99 E1-1D 0F 5B 00-5A 22 20 1A  ô¦¯SâGÖß?¤[ Z" ?
A7 1D F2 FC-67 28 40 54-3B 12 6C 97-78 4A B5 A2  °?=ng(@T;?lùxJ¦ó
3B 6C B7 29-21 56 B1 A3-1C F1 71 E9-D6 C3 FC FD  ;l+)!V¦ú?±qT++n²
F8 F1 45 E8-7B DD 67 63-FA 62 67 6A-EA 33 0C FB  °±EF{¦gc·bgjO3?v
8F 90 98 2F-11 39 65 64-A3 11 7C C1-38 29 67 0E  ÅÉÿ/?9edú?|-8)g?
```

After decryption

1. PNG Sig
2. dummy chunk start
3. chunk data (encrypted content of source file)
4. chunk crc
5. target file chunks
6. paddings

= target file
with an extra chunk at the beginning
+ padding

```
89 50 4E 47-0D 0A 1A 0A-00 00 00 80-61 61 61 61 ëPNG????   Çaaaa
B0 EC 40 7E-FB 1E 5D 0B-5D 87 A9 4A-AF A1 08 A8  ¦8@~v?]?]ç¬J»í?¿
9A D4 46 4A-75 87 6C 72-24 71 23 E6-66 AF 77 B7  Ü+FJuçlr$q#µf»w+
93 AC A7 B3-F5 81 CF C9-31 47 80 AA-73 43 9A C5  ô¼°¦)ü-+1GÇ¬sCÜ+
5A 0F 5F 40-C9 8B 4D AF-A0 D7 CD 3B-86 D0 58 32  Z¤_@+ïM»á+-;å-X2
E1 52 6A 36-E2 3E DD D5-5C 95 BB C5-8C 44 A5 8E  ßRj6G>¦+\ò++îDÑÄ
14 71 89 70-E2 25 F8 95-84 27 DD AD-E3 90 E9 50  ¶qëpG%°òä'¦¡pÉTP
C4 E7 20 FD-0E C6 4A 69-95 B6 0D 73-25 30 D9 9E  -t ²?¦Jiò¦?s%0+P
D1 01 42 A7-5E 32 18 85-A2 BD B8 61-19 9B 52 CF  -?B°^2?àó++a?¢R-
64 87 91 0E-00 00 00 0D-49 48 44 52-00 00 00 22 dçæ?   ?IHDR   "
00 00 00 1B-08 02 00 00-00 96 50 CA-F0 00 00 00      ???   ûP-=
01 73 52 47-42 00 AE CE-1C E9 00 00-00 06 62 4B  ?sRGB «+?T   ?bK
47 44 00 FF-00 FF 00 FF-A0 BD A7 93-00 00 00 09  GD       á+°ô   ?
70 48 59 73-00 00 0E C4-00 00 0E C4-01 95 2B 0E  pHYs  ?-  ?-?ò+?
1B 00 00 00-07 74 49 4D-45 07 DD 01-18 0C 39 2E  ?    •tIME•¦???9.
11 F1 8A 80-00 00 01 05-49 44 41 54-48 C7 BD 56  ?±èÇ  ??IDATH¦+V
CB 12 C3 20-08 04 C7 FF-FF 65 7A B0-43 09 8F 15  -?+ ??¦  ez¦C?Å§
EB 4C 38 29-59 40 61 21-B2 88 10 11-33 13 D1 5A  dL8)Y@a!¦ê??3?-Z
EB D6 8A 88-58 A5 22 1D-38 F5 20 22-9C DA BB A8  d+èêXÑ"?8) "£++¿
D6 52 F1 1D-A4 AE 39 F5-EE 6E 13 3D-62 64 8C 37  +R±?ñ«9)en?=bdî7
A9 16 67 B3-45 32 33 33-BB BC AD ED-AC 8A 01 24  ¬?g¦E233++¡f¼è?$
4D 54 0B 23-22 AA 4A ED-9D 52 8C 54-7E 1E 51 FB  MT?#"¬Jf¥RîT~?Qv
99 B9 91 59-5D B3 A2 5F-93 D0 CE E7-48 6B A3 9F  Ö¦æY]¦ó_ô-+tHkúƒ
AB 00 AA 01-48 BB 1E 55-33 82 B6 88-1E B7 DB 01  ½ ¬?H+?U3é¦ê?+¦?
68 D3 61 94-22 63 1A AD-C6 27 2D 66-A3 13 1E C0  h+aö"c?¡¦'-fú??+
BE FD 94 76-D3 FD 4C F3-F3 E9 3D 42-63 EE 62 4E  +²öv+²L==T=BcebN
9F 5D 31 9D-02 F2 14 8C-4C BF FE 2A-D2 A9 CD D1  ƒ]1¥?=¶îL+¦*-¬--
CC 4F 29 37-01 AF 2E CB-66 7D 8E A3-FE B0 2E AA  ¦O)7?».-f}Äú¦¦.¬
C1 91 6F D3-61 5C 05 6E-52 20 32 E8-25 42 53 F3  -æo+a\?nR 2F%BS=
87 11 95 00-19 7D A2 B7-40 87 54 5B-24 3A 66 E7  ç?ò ?}ó+@çT[$:ft
E0 47 CA 09-4A 07 B2 E7-5E 17 5B E4-F8 63 EC DF  aG-?J•¦t^?[S°c8¯
CE B4 34 C5-15 59 C1 81-56 CD 2C F2-03 4A 02 A6  +¦4+§Y-üV-,=?J?ª
B8 72 E2 63-1E 00 00 00-00 49 45 4E-44 AE 42 60  +rGc?    IEND«B`
82 0B 0B 0B-0B 0B 0B 0B-0B 0B 0B 0B-04 04 04 04  é???????????????
```

That was too easy :)

a more elegant solution ?

It works, but...

both files aren’t standard

appended data is a giveaway

A smarter appended data

since we have to handle the file format

To prevent obvious appended data

- hide ‘external’ data just after the source data
    - provided the extra data is ignored
- combine encryption/decryption block

Appended data

at file level:

- original file
- *appended data*

Appended data on known format

if we know the structure, this gives:

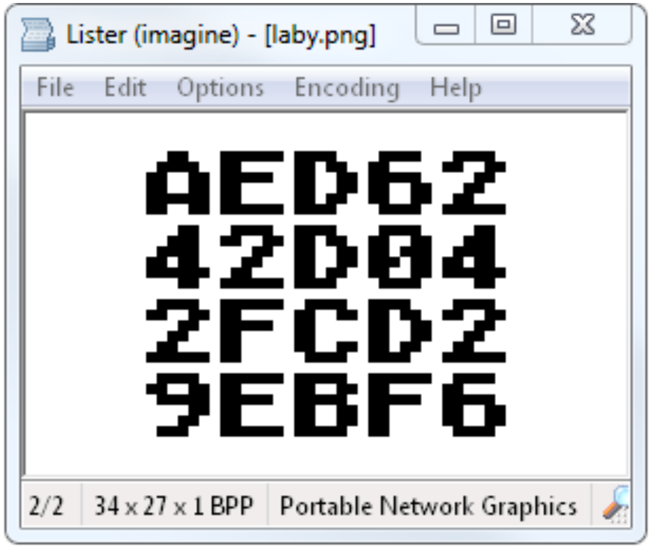
- original file
  - header
  - format-specific data
  - footer
- *appended data*

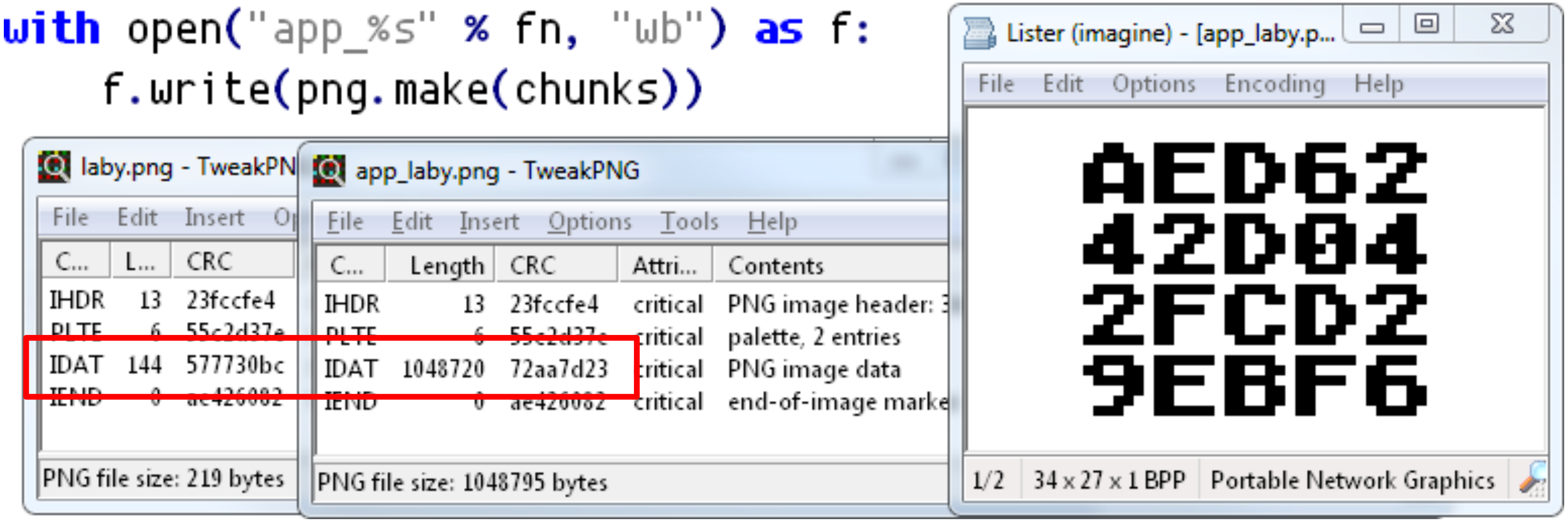
Append data *in* the format

right after the original dat

- original file
  - header
  - format-specific data
    - *appended data*
  - footer

```
import sys, png, os

fn = sys.argv[1]

with open(fn, "rb") as f:
    chunks = png.read(f)

for chunk in chunks:
    if chunk[0] == "IDAT":
        chunk[1] += os.urandom(1024 * 1024)

with open("app_%s" % fn, "wb") as f:
    f.write(png.make(chunks))
```


Lister (imagine) - [laby.png]
File Edit Options Encoding Help
AED62
42D04
2FCD2
9EBF6
2/2
34 x 27 x 1 BPP
Portable Network Graphics


laby.png - TweakPN
File Edit Insert O
app_laby.png - TweakPNG
File Edit Insert Options Tools Help
Lister (imagine) - [app_laby.p...
File Edit Options Encoding Help
AED62
42D04
2FCD2
9EBF6
1/2
34 x 27 x 1 BPP
Portable Network Graphics

C...	L...	CRC
IHDR	13	23fccfe4
PLTE	6	55c2d37e
IDAT	144	577730bc
IEND	0	ae426082

PNG file size: 219 bytes

C...	Length	CRC	Attri...	Contents
IHDR	13	23fccfe4	critical	PNG image header: 3
PLTE	6	55c2d37e	critical	palette, 2 entries
IDAT	1048720	72aa7d23	critical	PNG image data
IEND	0	ae426082	critical	end-of-image marke

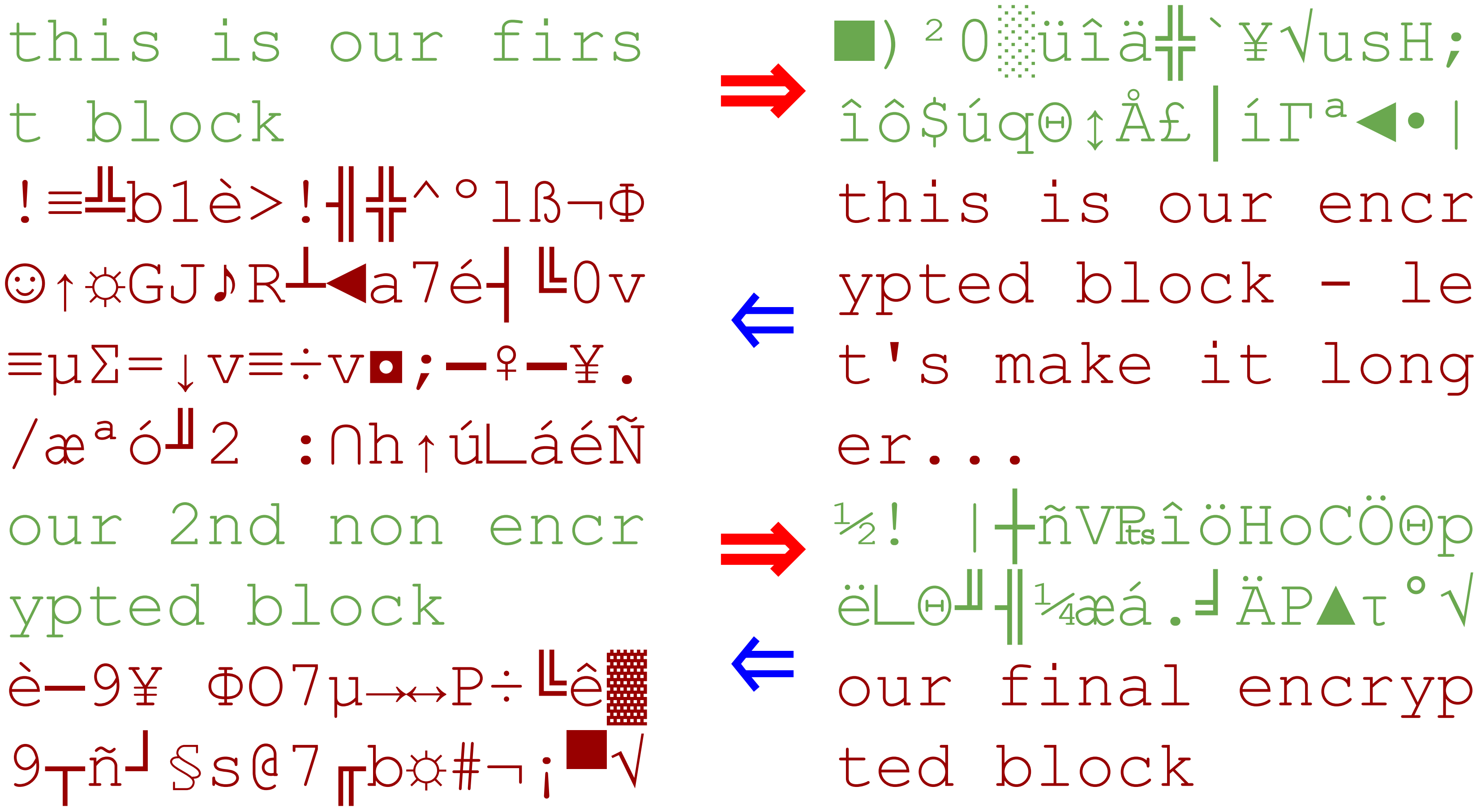
PNG file size: 1048795 bytes

appending data at file format level

Combining blocks

since blocks encryption/decryption only depends on ***previous*** blocks & parameters

1. append data
2. perform operation on the whole block
   - alternate encryption and decryption
3. repeat

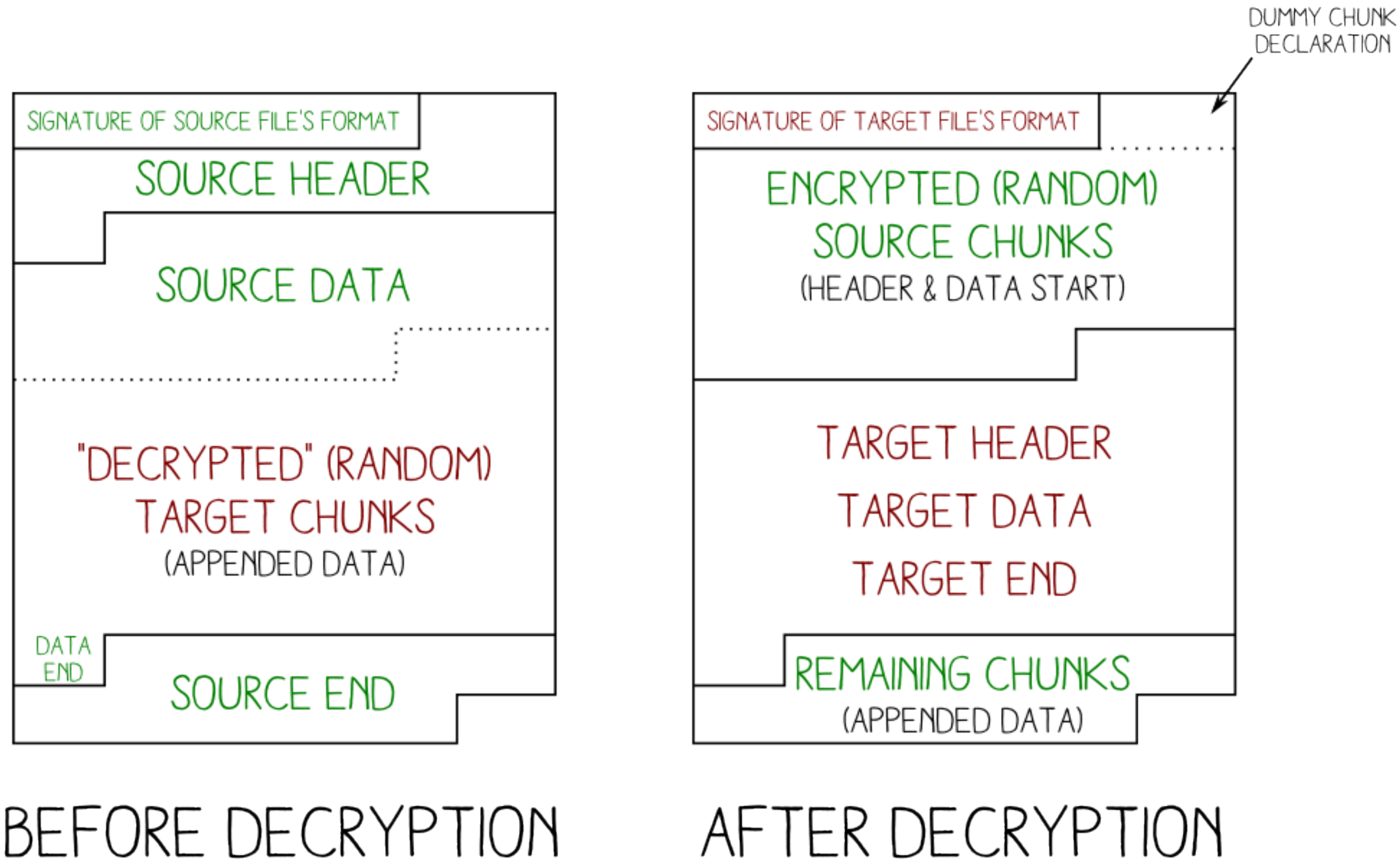


```
this is our firs
t block
!≡╨b1è>!╢╬^°1ß¬Φ
☺↑☼GJ♪R┴◄a7é┤╙0v
≡µΣ=↓v≡÷v◘;▬♀▬¥.
/æªó╜2 :∩h↑ú└áéÑ
our 2nd non encr
ypted block
è▬9¥ Φ07µ→↔P÷╙ê▓
9┬ñ┘§s@7╓b☼#¬¡■√
```

```
■)²0░üîä╬`¥√usH;
îô$úqΘ↕Å£│íΓª◄•|
this is our encr
ypted block - le
t's make it long
er...
½! |┼ñV₧îöHoCÖΘp
ë└Θ╜╢¼æá.╛ÄP▲τ°√
our final encryp
ted block
```

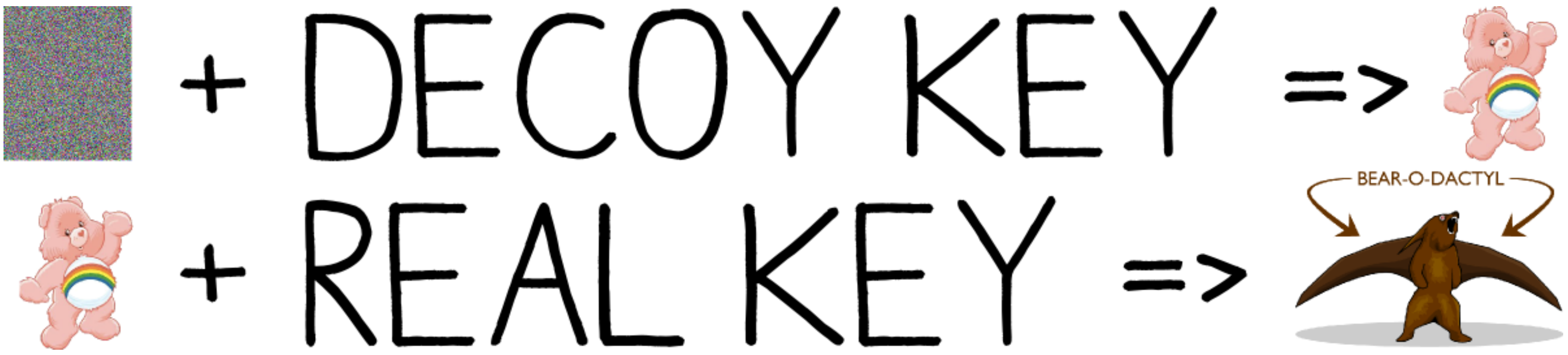
chaining encrypted & decrypted block

key = "alsmotrandomkey!" IV = "Initialization.."



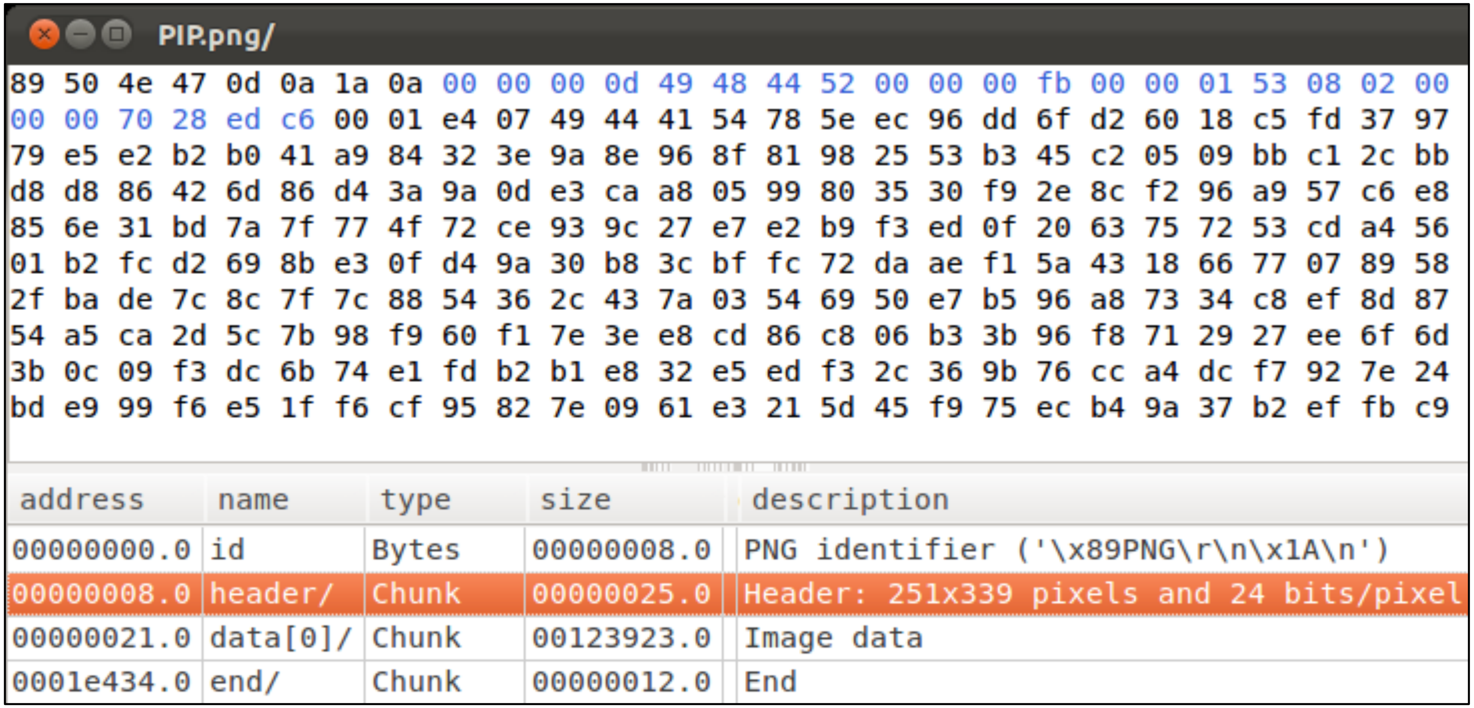
a more complex layout

→ the ‘start’ file is a standard PNG



+ DECOY KEY =>

+ REAL KEY =>



```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 00 fb 00 00 01 53 08 02 00
00 00 70 28 ed c6 00 01 e4 07 49 44 41 54 78 5e ec 96 dd 6f d2 60 18 c5 fd 37 97
79 e5 e2 b2 b0 41 a9 84 32 3e 9a 8e 96 8f 81 98 25 53 b3 45 c2 05 09 bb c1 2c bb
d8 d8 86 42 6d 86 d4 3a 9a 0d e3 ca a8 05 99 80 35 30 f9 2e 8c f2 96 a9 57 c6 e8
85 6e 31 bd 7a 7f 77 4f 72 ce 93 9c 27 e7 e2 b9 f3 ed 0f 20 63 75 72 53 cd a4 56
01 b2 fc d2 69 8b e3 0f d4 9a 30 b8 3c bf fc 72 da ae f1 5a 43 18 66 77 07 89 58
2f ba de 7c 8c 7f 7c 88 54 36 2c 43 7a 03 54 69 50 e7 b5 96 a8 73 34 c8 ef 8d 87
54 a5 ca 2d 5c 7b 98 f9 60 f1 7e 3e e8 cd 86 c8 06 b3 3b 96 f8 71 29 27 ee 6f 6d
3b 0c 09 f3 dc 6b 74 e1 fd b2 b1 e8 32 e5 ed f3 2c 36 9b 76 cc a4 dc f7 92 7e 24
bd e9 99 f6 e5 1f f6 cf 95 82 7e 09 61 e3 21 5d 45 f9 75 ec b4 9a 37 b2 ef fb c9
```

address	name	type	size	description
00000000.0	id	Bytes	00000008.0	PNG identifier ('\x89PNG\r\n\x1A\n')
00000008.0	header/	Chunk	00000025.0	Header: 251x339 pixels and 24 bits/pixel
00000021.0	data[0]/	Chunk	00123923.0	Image data
0001e434.0	end/	Chunk	00000012.0	End

a PNG encrypted in a standard PNG

a note on ZIP

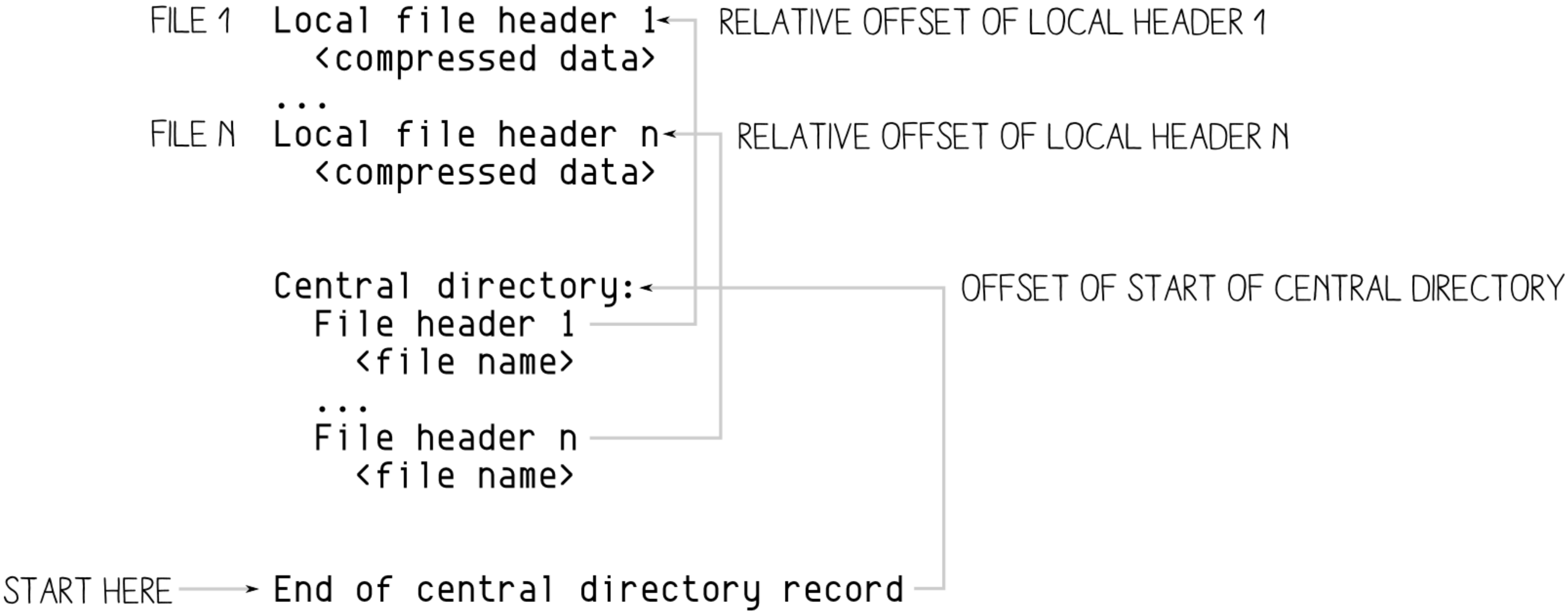
it’s not as permissive as we usually think

ZIP file, in practice

- the signature is not enforced at offset 0

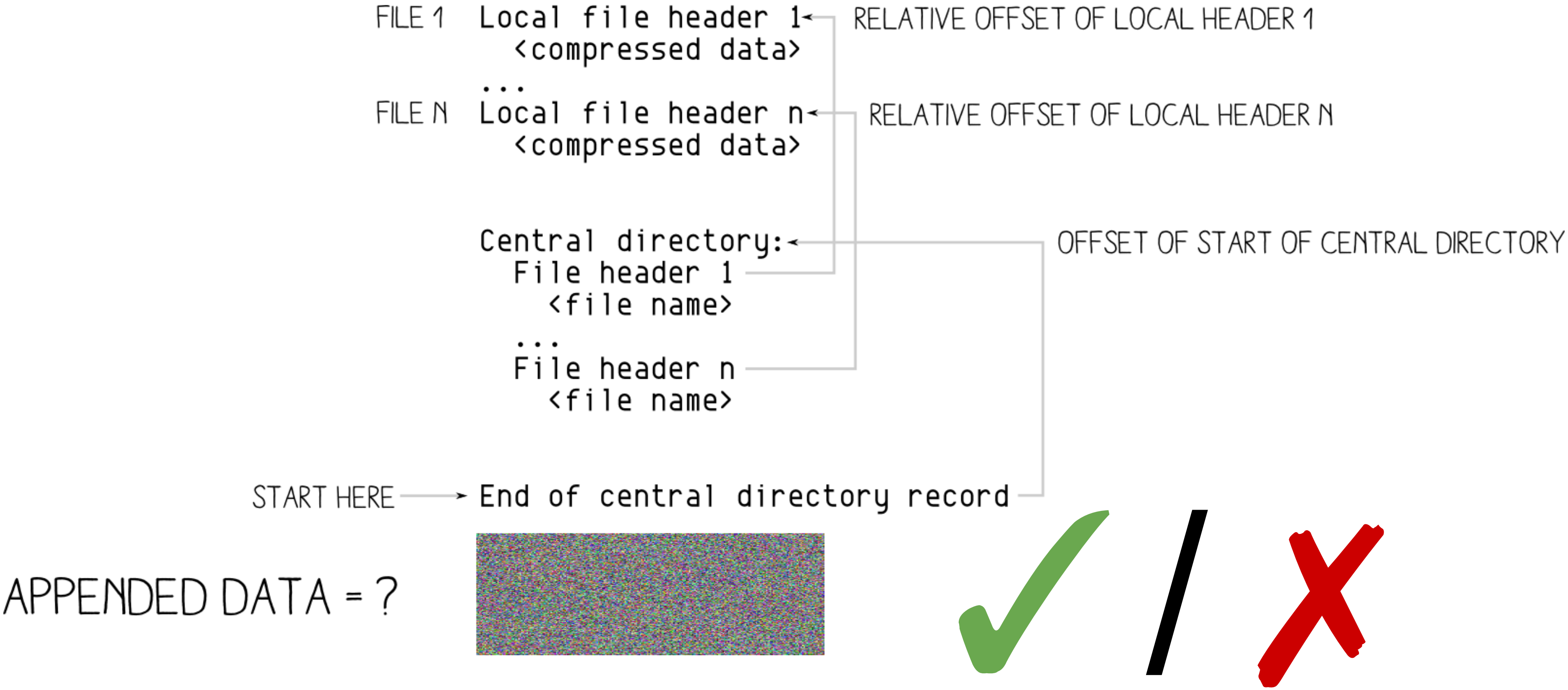
⇒ ZIP data is usually remembered
as ‘valid anywhere’ in the file.

That’s wrong:
ZIP is different from modern standards,
but it doesn’t work ‘anywhere’



```
FILE 1   Local file header 1    RELATIVE OFFSET OF LOCAL HEADER 1
           <compressed data>
         ...
FILE N   Local file header n    RELATIVE OFFSET OF LOCAL HEADER N
           <compressed data>


         Central directory:     OFFSET OF START OF CENTRAL DIRECTORY
           File header 1
             <file name>
           ...
           File header n
             <file name>


START HERE → End of central directory record
```

ZIP is parsed backward



```
FILE 1   Local file header 1
           <compressed data>
         ...
FILE N   Local file header n
           <compressed data>

         Central directory:
           File header 1
             <file name>
           ...
           File header n
             <file name>

START HERE → End of central directory record
```

RELATIVE OFFSET OF LOCAL HEADER 1

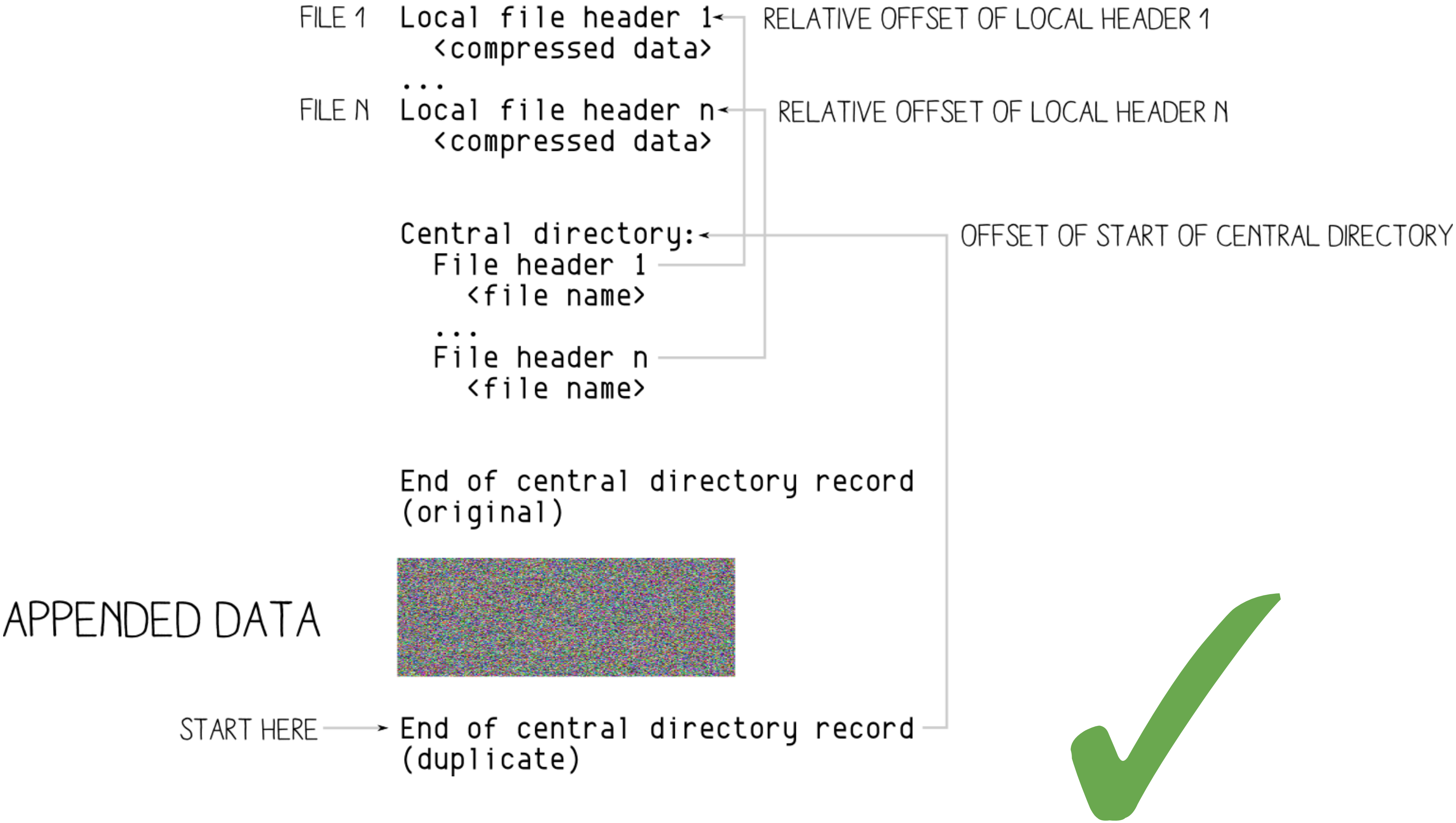
RELATIVE OFFSET OF LOCAL HEADER N

OFFSET OF START OF CENTRAL DIRECTORY

APPENDED DATA = ?

✓ / ✗

Tools don’t accept too much appended data size



FILE 1 Local file header 1 ← RELATIVE OFFSET OF LOCAL HEADER 1

<compressed data>

...

FILE N Local file header n ← RELATIVE OFFSET OF LOCAL HEADER N

<compressed data>

Central directory: ← OFFSET OF START OF CENTRAL DIRECTORY

File header 1

<file name>

...

File header n

<file name>

End of central directory record (original)

APPENDED DATA

START HERE → End of central directory record (duplicate)

duplicating the End of Central Directory increases compatibility

Increase ZIP compatibility

Duplicate EoCD after appended data

(cheap internal appended data)

⇒ tools will parse the ZIP correctly

⇒ AES(PNG) = APK

How Android Malware Fights
(and we fight back!)
or Research on obfuscation in Android malware

Axelle Apvrille, Ruchna Nigam
Fortiguard Labs, Fortinet

CARO Workshop, Melbourne, Florida, USA
May 2014

GynCryption

as suggested by Gynvael Coldwind

- JPG only requires 4 bytes

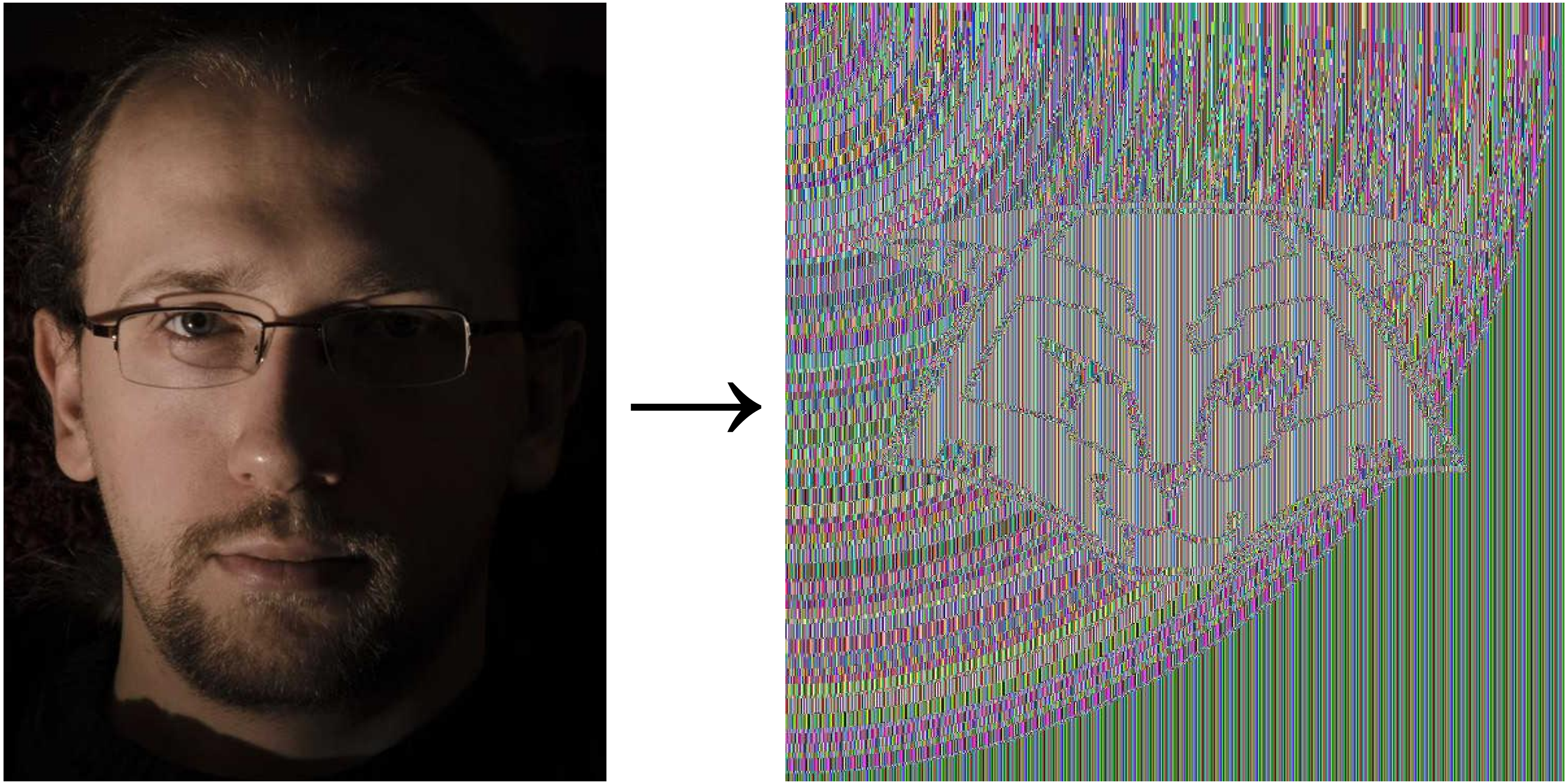
⇒ use ECB and bruteforce the key

recompress the JPG if the chunk size is too big

- ○ the chunk size is ‘random’ but stored on 2 bytes
- ○ same dimensions ⇒ same 1st block

Steps

1. get P1
2. bruteforce key
   until C1 starts with `FF D8 FF FE`
   (required ~18M iterations for me)
3. shrink S if bigger than chunk’s size
4. pad S until the right offset
5. encrypt S
6. append T
   - minus its signature
7. decrypt



PoC

Source & PoCs

http://corkami.googlecode.com/svn/trunk/src/angecryption/

Conclusion

- a funny trick
  - a bit of crypto magic, a bit of binary magic
  - having fun with usually scary topics
- steganographic application
- a reminder that:
  - crypto is not always ‘random’
  - binary manipulation doesn’t require full understanding

possible applications:

- protocols: JWE, OCSP...

Suggestions?

- challenging formats
- applications
- unforeseen consequences

ACK

@veorq

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

@angealbertini

corkami.com