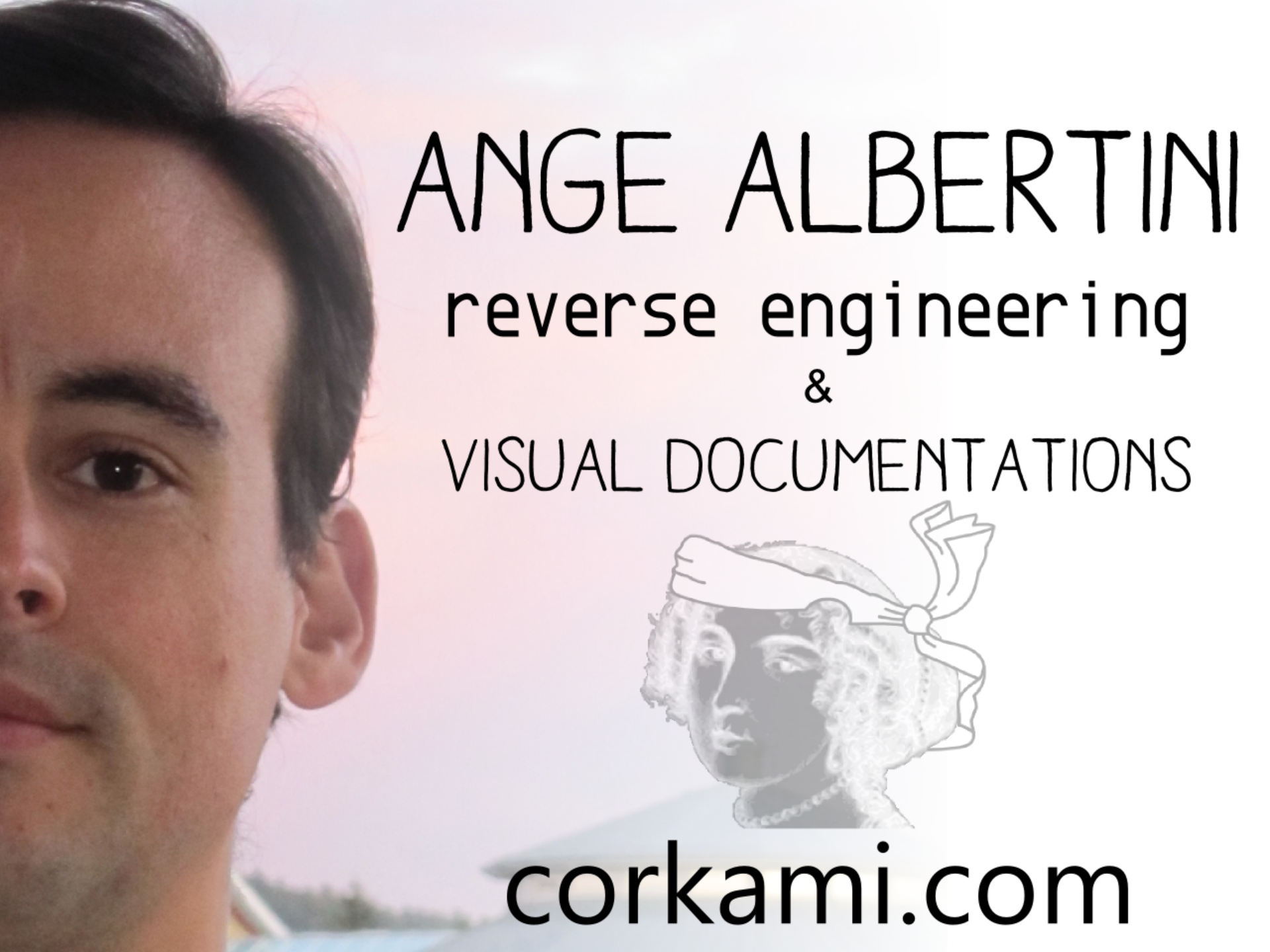

JOUE
À LA
CAYAC !


ANGE ALBERTINI
reverse engineering
&
VISUAL DOCUMENTATIONS
corkami.com

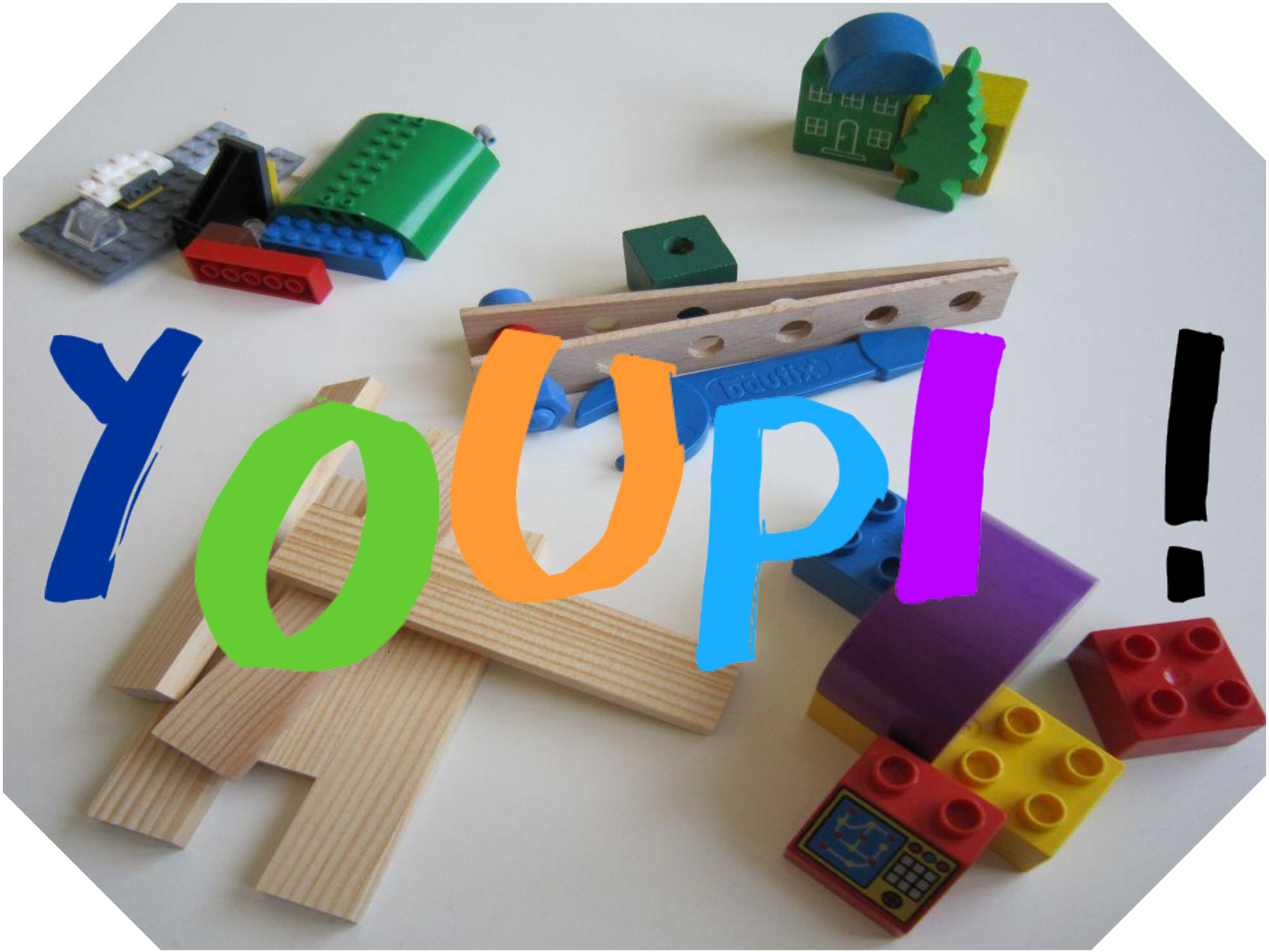
$$
\begin{aligned}
{\tau'}_F^{\overline{\beta}} &= |m - 2H(\overline{\beta})| \\
&\quad - \frac{1}{2^n(2^n-1)} \sum_{a \in \mathbb{F}_2^{n*}} \left| \sum_{j=1}^{m} \left((-1)^{\overline{\beta}_j} \mathcal{A}_{F_j}(a) + \sum_{i=1, i \neq j}^{m} (-1)^{\overline{\beta}_i} \mathcal{C}_{F_i, F_j}(a) \right)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

La crypto, c’est compliqué.

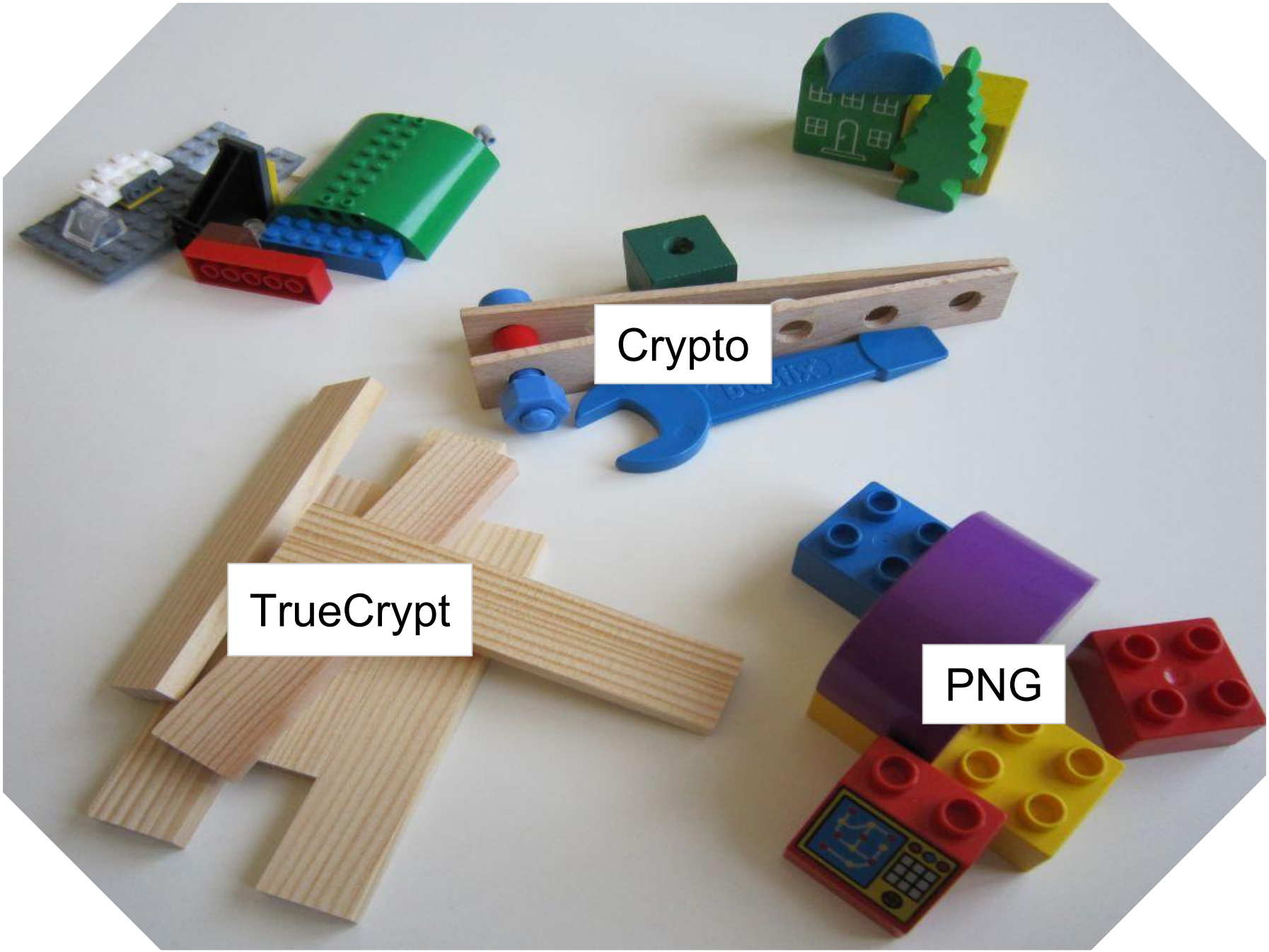
$$
\begin{aligned}
{\tau'}_F^{\overline{\beta}} &= |m - 2H(\overline{\beta})| \\
&\quad - \frac{1}{2^n(2^n-1)} \sum_{a \in \mathbb{F}_2^{n*}} \left| \sum_{j=1}^{m} \left((-1)^{\overline{\beta}_j} \mathcal{A}_{F_j}(a) + \sum_{i=1, i \neq j}^{m} (-1)^{\overline{\beta}_i} \mathcal{C}_{F_i,F_j}(a) \right)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

HORREUR

Ma réaction habituelle...



Ça ne m’empêche pas de jouer avec.



Jouons donc un peu...

on dit “chiffrer” pas “crypter”

“décrypte
le cipher”

AES

Advanced Encryption Standard

1 bloc (16 octets)

1 bloc (16 octets)
+
1 clé (16 octets)*

dorénavant, on dira AES pour AES-128

1 bloc (16 octets)

+

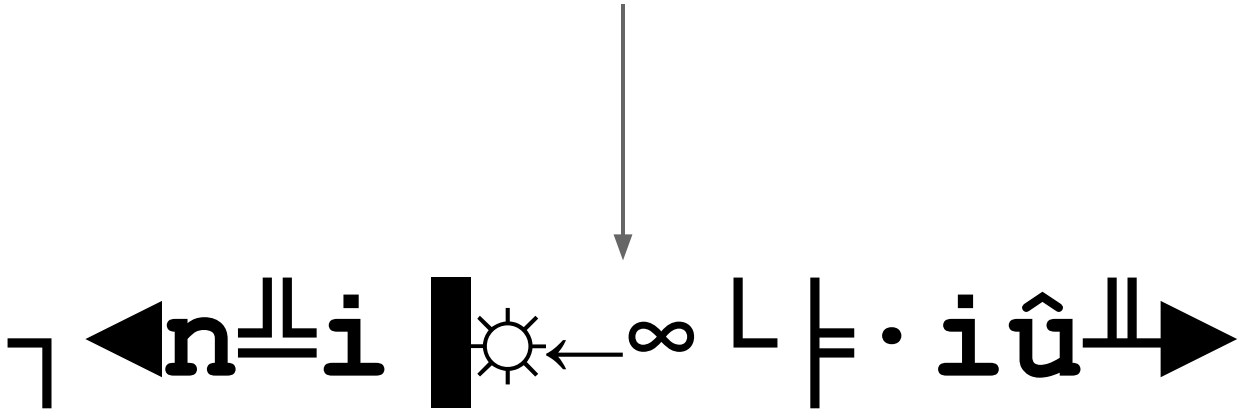
1 clé (16 octets)

1 bloc (16 octets)

a block of text.

+

MySecretKey12345



(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)

Le moindre changement de clé ou de bloc donne un résultat complètement différent

on ne peut pas contrôler la sortie

les différences sont imprévisibles

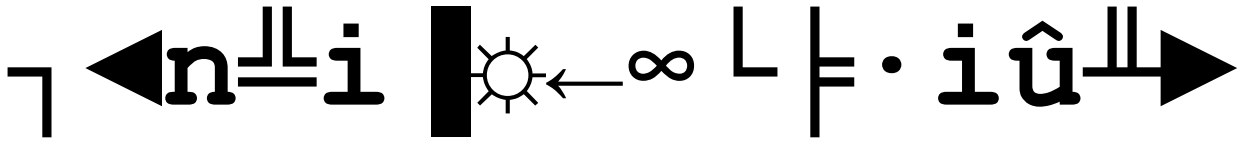
l’opération inverse

a block of text.

+

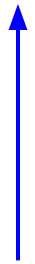
MySecretKey12345

chiffrement



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

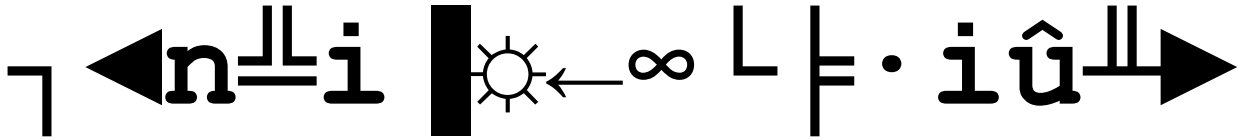
a block of text.



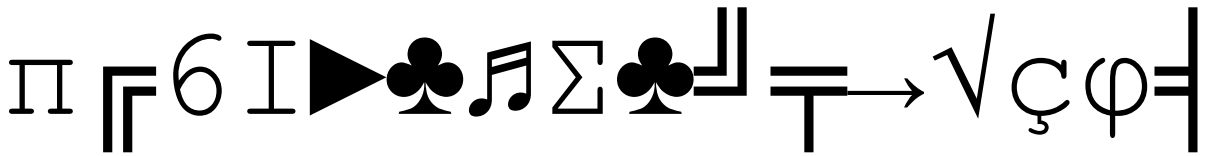
déchiffrement

MySecretKey12345

+



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

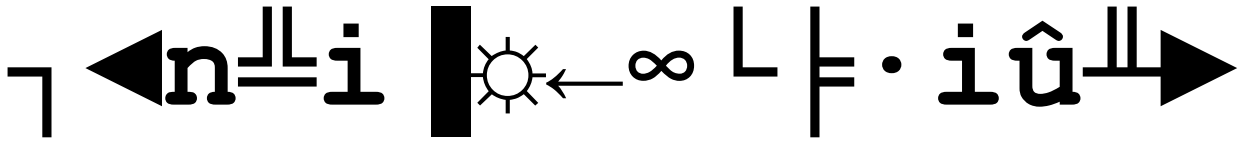


(E3 C9 36 49 10 05 0E E4 05 BC D1 1A FB 87 ED B5)

déchiffrement

MySecretKey12346

+



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

avec la clé de chiffrement,
on peut retrouver
le texte source

sans la clé de chiffrement,
on ne peut rien faire
avec le bloc chiffré

“en clair” et “chiffré” ne sont que des appellations

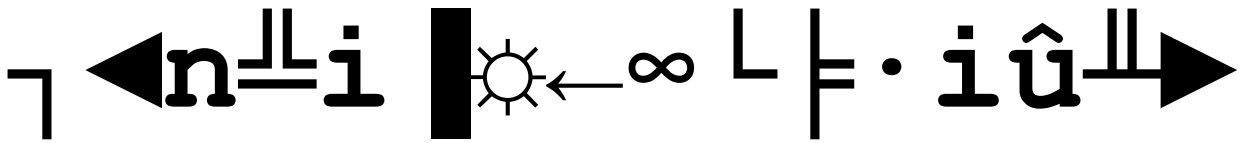
chiffrement ⇔ déchiffrement
ne sont que des fonctions inverses

a block of text.

+

MySecretKey12345

chiffrement

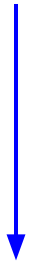


(BF 11 6E CA 69 DE 0F 1B EC C0 C6 F9 69 96 D0 10)

a block of text.

+

MySecretKey12345



déchiffrement

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.

MySecretKey12345

+

ä/ë-╤╥7 ↓h|☻⌂µ[←Ñ

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

on peut déchiffrer du texte en clair

on retrouve le texte original après chiffrement
⇒ on peut contrôler le résultat d'un chiffrement

Récapitulons

- AES chiffre un bloc
  - on ne contrôle pas le résultat
- le bloc chiffré peut être restauré
  - avec la clé de chiffrement
- chiffrer⇔déchiffrer ne sont que des inverses
  - on peut déchiffrer du texte en clair
  - on le retrouve après chiffrement
- on ne peut pas contrôler l'entrée *et* la sortie
  - l'une, ou l'autre

PNG

Portable Network Graphics

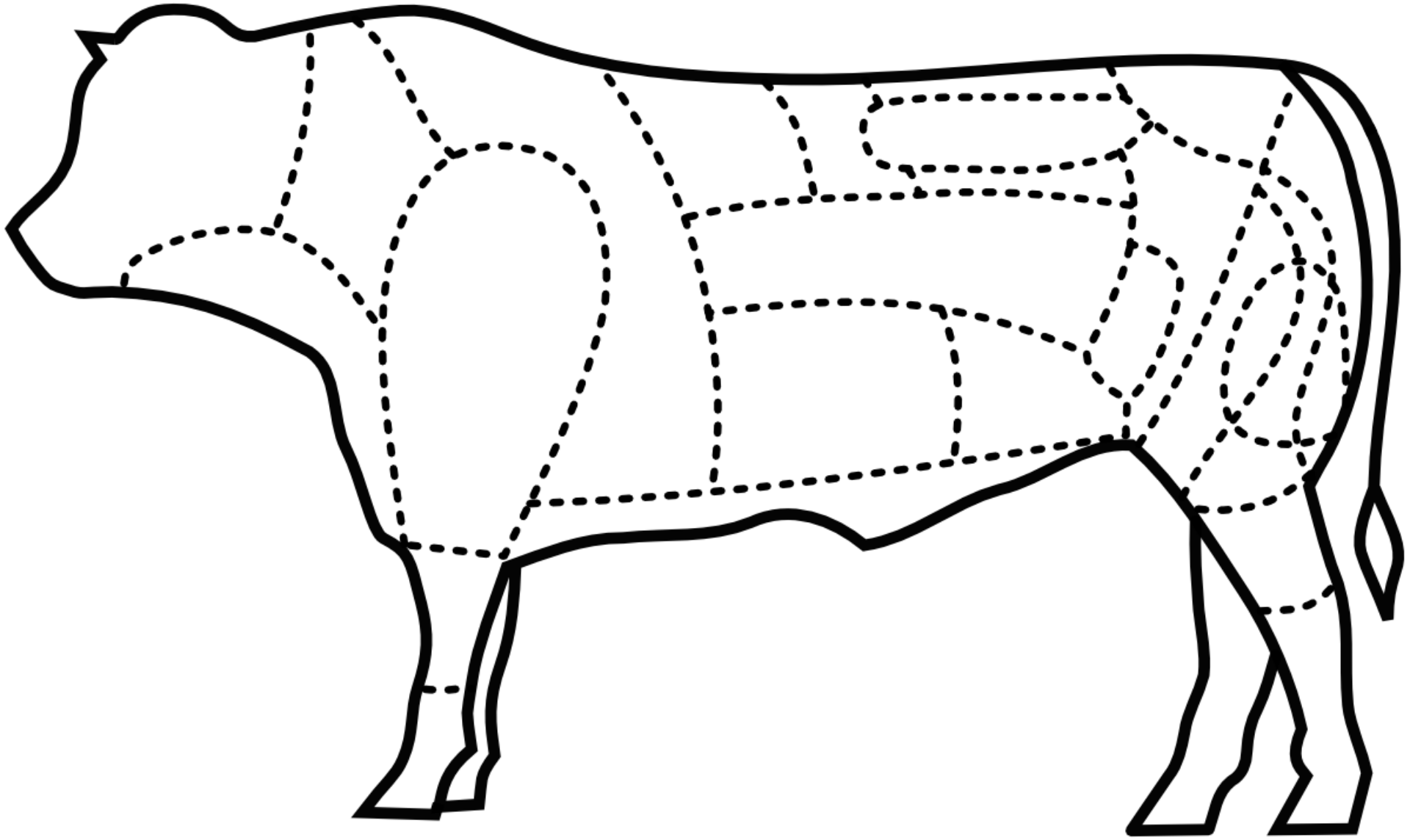
PiNG

Portable Network Graphics

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

SHA-1 349841408d1aa1f5a8892686fbdf54777afc0b2c

Prenons un fichier, que vous avez peut-être déjà vu.

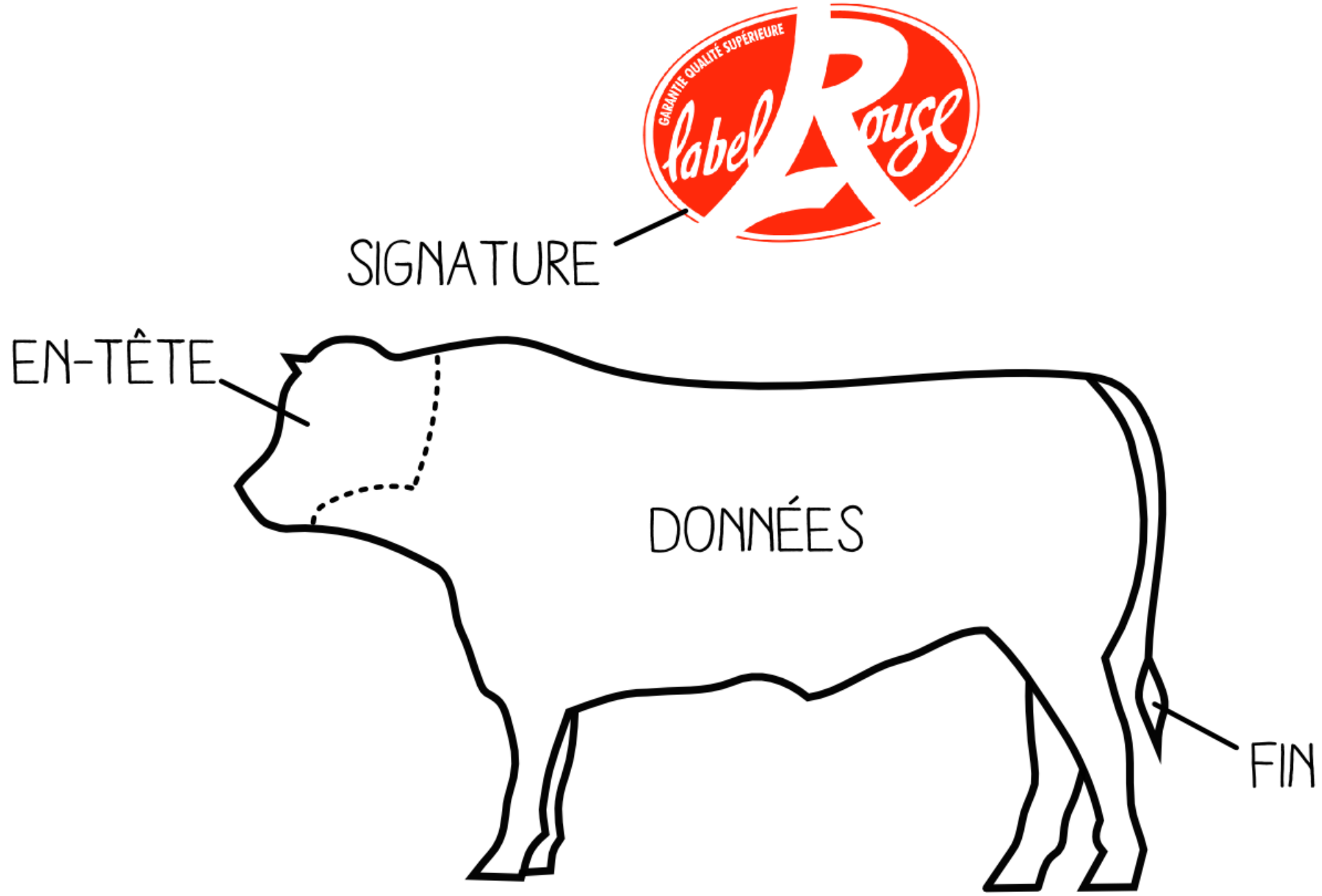


Le format PNG se découpe en pièces (chunks), comme un boeuf.

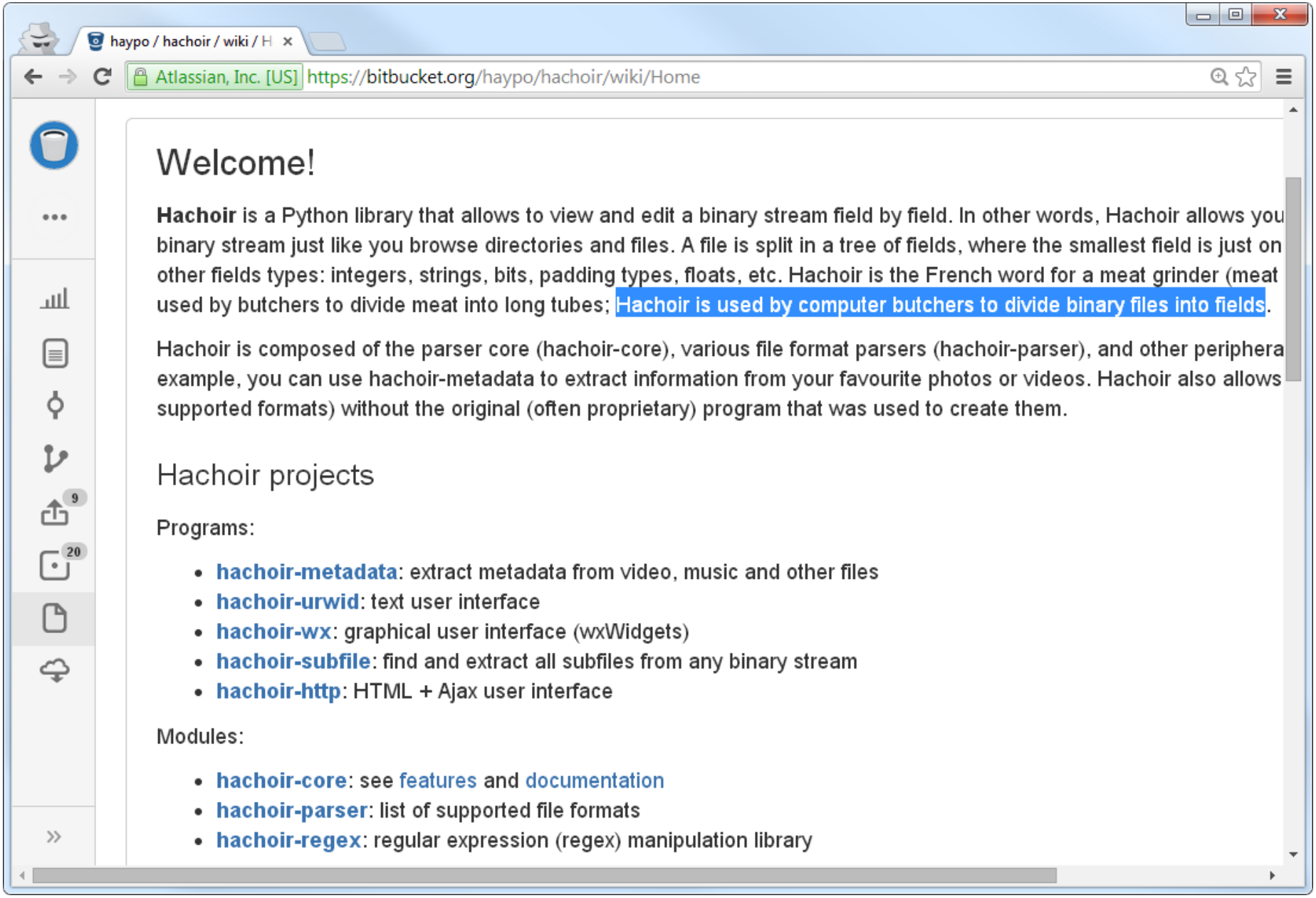
Chunk

- Fichier structuré en pièces de taille variable
  - critiques ou auxiliaires
- Structure commune
  - Indépendante du contenu et de son interprétation

⇒ Stocker des informations propriétaires en garantissant une compatibilité minimale



Notre boeuf décomposé.



Notre outil pour bouchers informatiques.

```
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

Une signature, puis une succession de chunks.

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

Signature obligatoire en position `0`

- identifie le fichier
- identifie les erreurs de transfert
  - \x89 : non ASCII (ASCII = [0 - 128])
  - \r\n puis \n : fins de lignes de standards différents
  - ^Z (\x1A) : “Fin De Fichier”

Chunk

- Structure commune:
  a. taille, sur 4 octets
  b. type, de 4 lettres
     - 1ère lettre minuscule = chunk auxiliaire
  c. données
  d. somme de contrôle
     - CRC32(type + données)
- on peut ajouter des chunks personnalisés

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
```

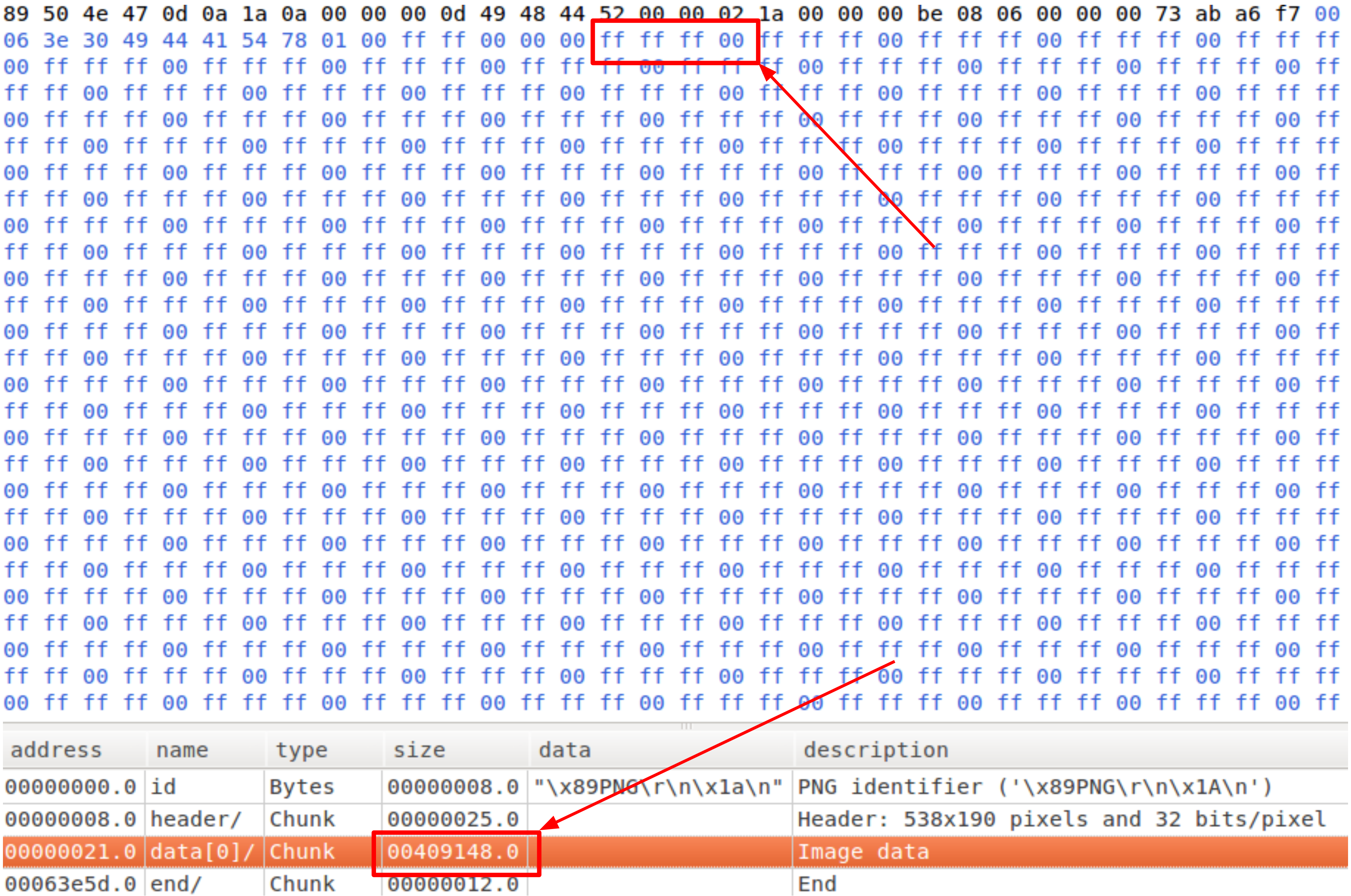
address	name	type	size	data	description
	../				
00000008.0	size	UInt32	00000004.0	13	Size
0000000c.0	tag	FixedString<ASCII>	00000004.0	"IHDR"	Tag
00000010.0	width	UInt32	00000004.0	538	Width (pixels)
00000014.0	height	UInt32	00000004.0	190	Height (pixels)
00000018.0	bit_depth	UInt8	00000001.0	8	Bit 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

Chunk IHDR : en-tête contenant des informations sur l’image

```
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

Chunk IDAT (compressé): valeurs des pixels



IDAT, une fois décompressé

(FF FF FF 00 = noir 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

Chunk IEND : Fin (de structure) du fichier

```
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>\...	

Ce qui vient après le chunk IEND est ignoré.

Récapitulons

Structure:

1. Signature en position `0`
2. Succession de chunks
   a. `IHDR` en-tête
   b. `IDAT` données de l'image
   c. `IEND` fin de l'image

“Google, je sais comment ça marche!”

Toi aussi, épate tes amis.

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)

Le chiffrement casse la 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
```

Sans signature, le fichier chiffré n’est pas valide.

Si on chiffre un PNG, on n’obtient pas un PNG

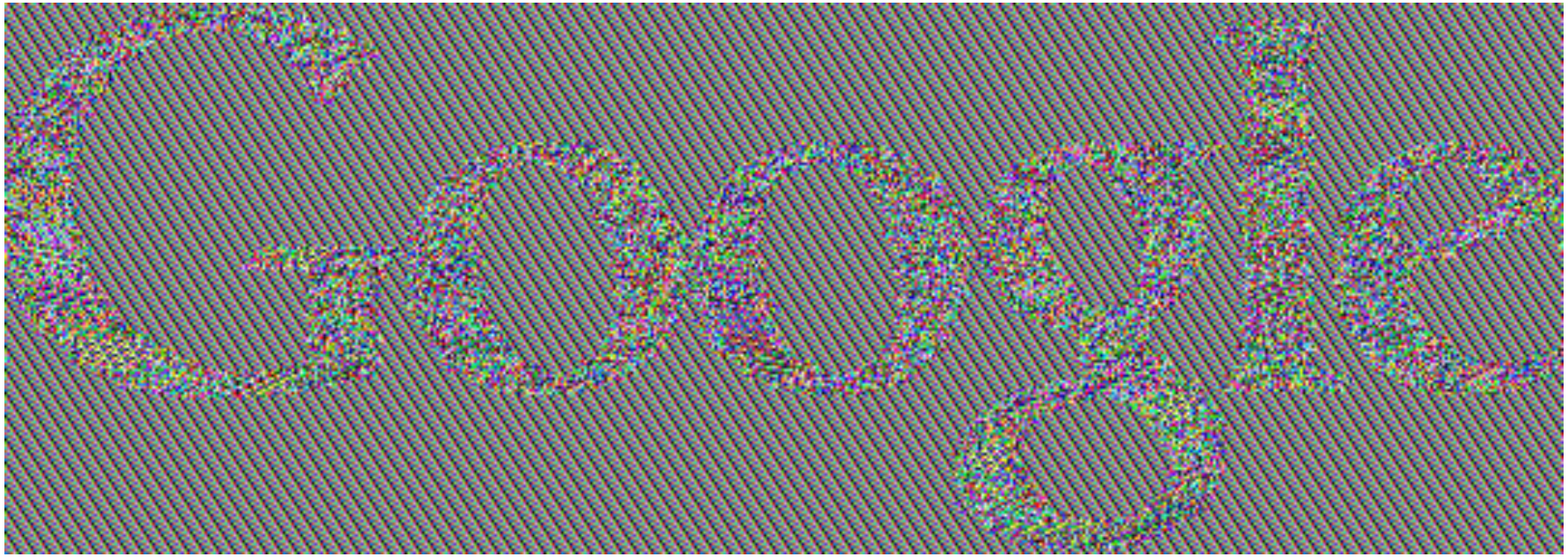
la signature est cassée, la structure aussi (***a priori***)

Comment faire pour chiffrer Google en  ?

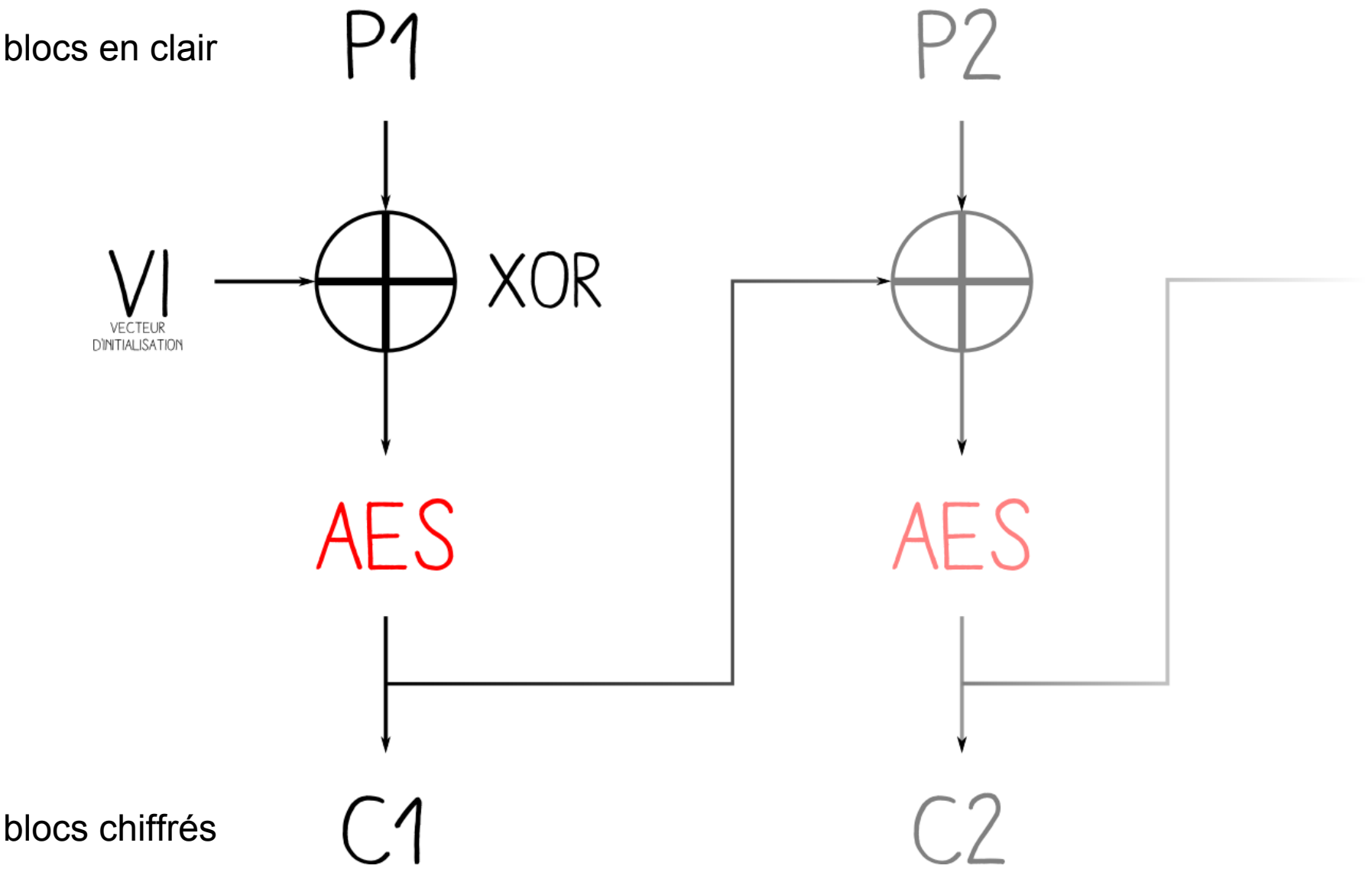
Comment contrôler entrée *et* sortie ?

AES opère sur un bloc

Comment l’appliquer à un fichier?



Ce qu’on obtient si on chiffre chaque bloc indépendamment (mode ECB)



En mode CBC, chaque chiffrement dépend des blocs précédents

Le mode CBC

- “enchaînement des blocs”
  - considéré comme sûr
  - implicite, dorénavant
  - le chiffrement dépend des blocs *précédents*
- utilise un vecteur d’initialisation
  - paramètre supplémentaire
  - arbitraire
  - en pratique, il devrait être imprévisible

clé

+

Vecteur d’Initialisation

X blocs

X blocs

Relations entre blocs et VI

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

* La clé utilisée est toujours la même.

Relations entre blocs et VI

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

$$DEC(C1) = DEC(ENC(VI \wedge P1))$$

on déchiffre chaque coté

Relations entre blocs et VI

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

$$DEC(C1) = DEC(ENC(VI \wedge P1))$$

ça s'annule

Relations entre blocs et VI

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

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

Relations entre blocs et VI

$$C1 = ENC(VI \oplus P1)$$

$$DEC(C1) = VI \oplus P1$$

$$P1 \oplus DEC(C1) = VI \oplus P1 \oplus P1$$

on applique un XOR sur chaque coté

Relations entre blocs et VI

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

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

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

ça s’annule

Relations entre blocs et VI

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

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

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

Relations entre blocs et VI

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

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

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

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

on obtient VI à partir de P1 et C1

VI = P1 ^ DEC(C1)

P1, C1 sont les 16 premiers octets de nos fichiers source et cible

une fois la clé ***c*** choisie,

on déchiffre C1

on applique un XOR avec P1

on obtient le VI

c **clé**

+

VI **Vecteur d'Initialisation**

Px **X blocs**

Cx **X blocs**

c

VI

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)

c IVManipulation!!

VI

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)

c IVManipulation!!

VI 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)

c IVManipulation!!

VI 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)

c **IVManipulation!!**

\+

VI **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** ✓

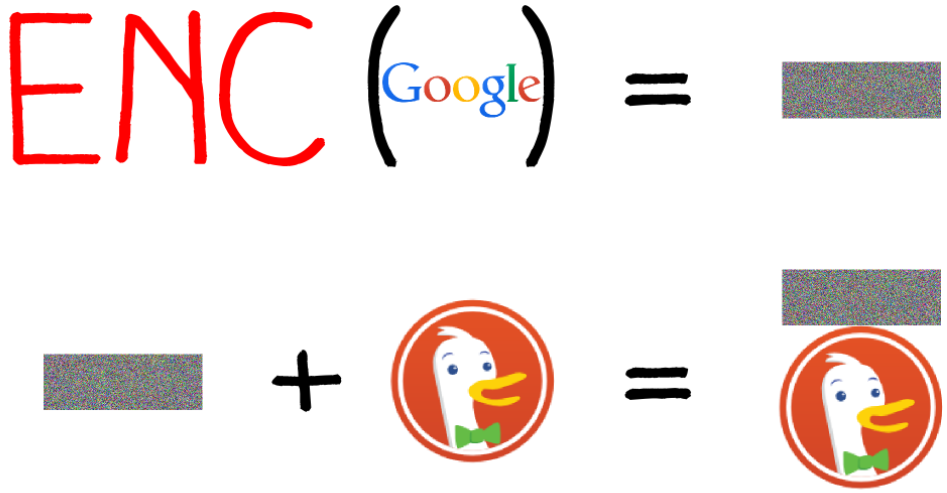
(89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52)

Status

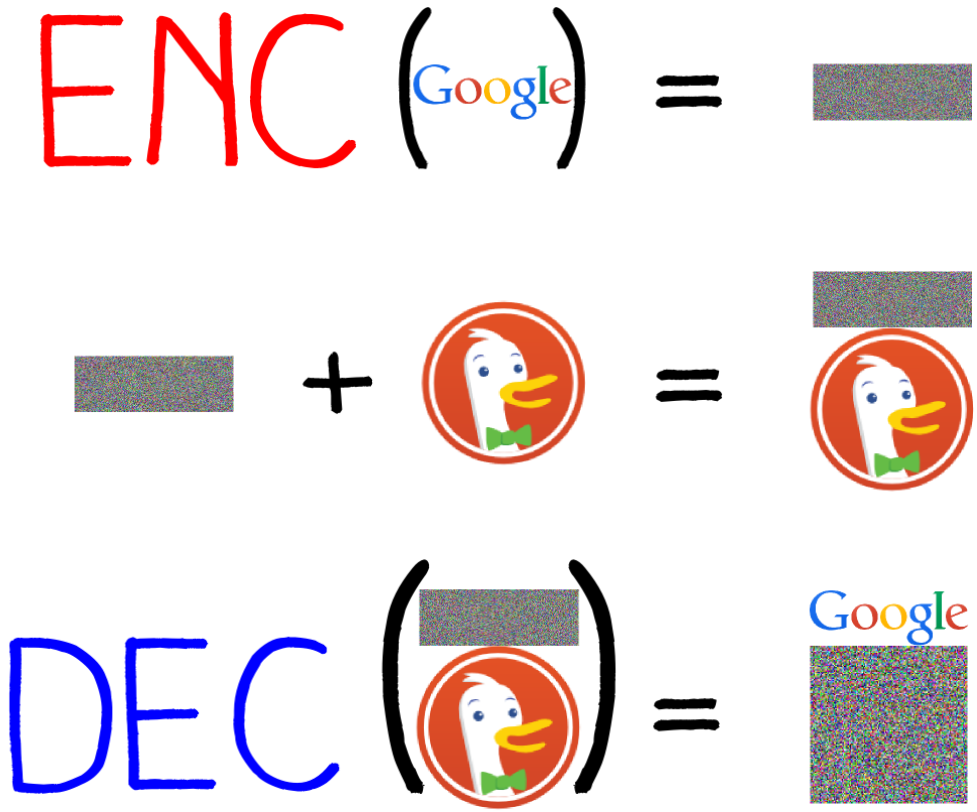
- on contrôle le premier bloc chiffré
  - on a une signature valide
- on ne contrôle rien d’autre
  - pas de structure valide

Comment contrôler la structure après chiffrement ?

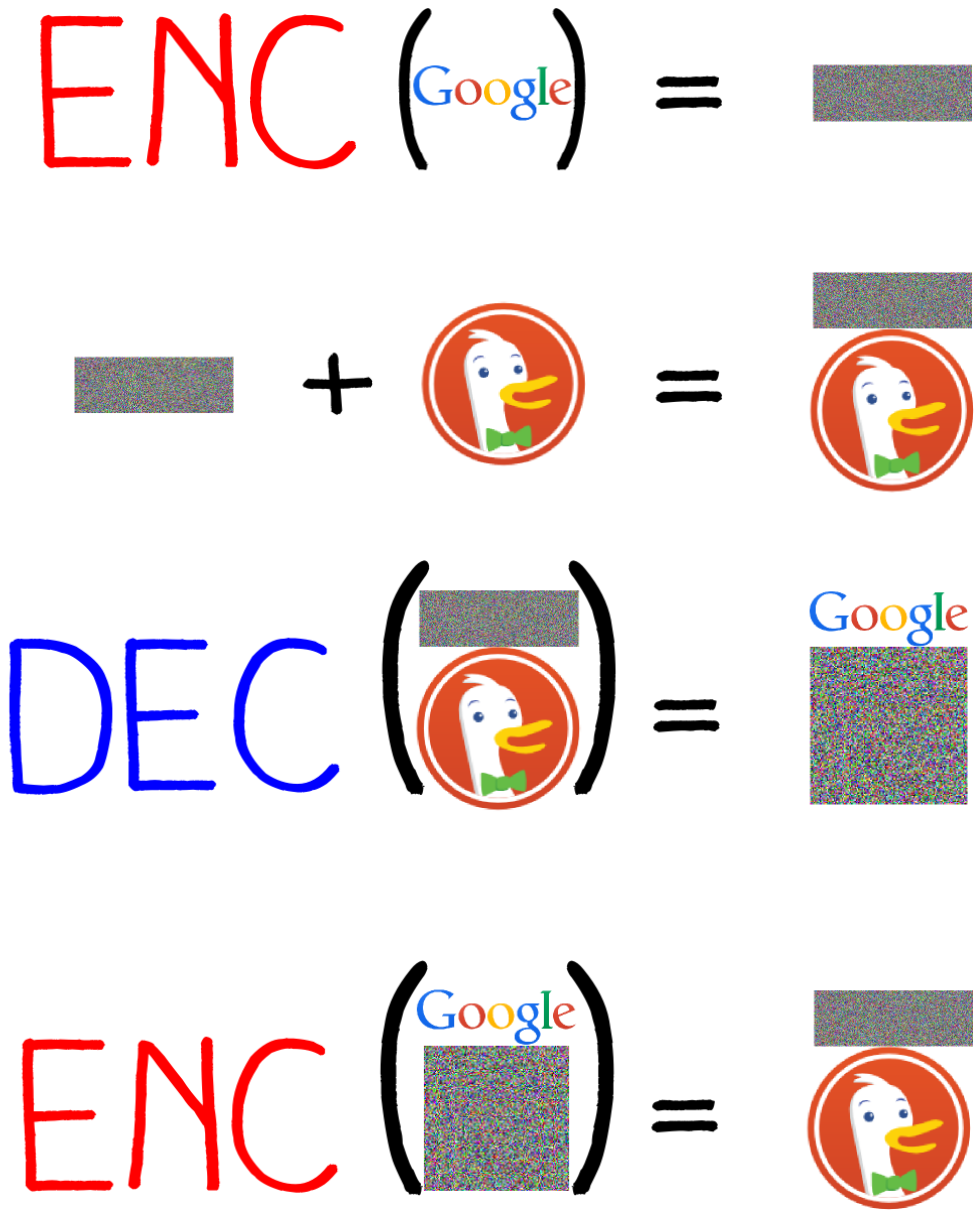
Si on chiffre notre image, on obtient un fichier aléatoire.



Si on ajoute du contenu en clair après ce bloc aléatoire...



… on peut le déchiffrer et retrouver notre image source.
(suivie d'un autre bloc aléatoire)



En chiffrant le résultat, on obtient le premier bloc aléatoire suivi de notre image cible.

Pré-déchiffrer des données

- Pré-déchiffrer les chunks de la cible
- Les ajouter à la fin du fichier source
  - dans le bloc suivant (*bourrage* si nécessaire)
  - qui est toujours valide grâce au marqueur `IEND`

Status

- On contrôle
  - un peu de l’entrée
  - un peu de la sortie
- Le fichier source est encore valide
  - fichier original (valide),
  - suivi de données déchiffrées

Comment contrôler les données chiffrées ?

On fera sans ☺

On va demander au format de les ignorer

```
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

Ajout d’un chunk standard de commentaire (de type ‘tEXt’)

```
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

La position du chunk n’importe pas.

```
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

Création d’un chunk ‘bing’ entièrement personnalisé

www.w3.org/TR/PNG/#11IHDR

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	

L'en-tête *devrait* être en premier.
En pratique, ça n'a pas d'importance (sauf pour Hachoir)

Récapitulons

Ajout de chunks auxiliaires:

- *type* en minuscules
- ordre des chunks sans grande importance

⇒ on peut rajouter n'importe quelle donnée dans un chunk supplémentaire

créer un chunk auxiliaire pour *couvrir* les données chiffrées

⇒ elles seront ignorées

⇒ le fichier chiffré sera valide !

Status

on contrôle après chiffrement:

- le premier bloc
  - la signature (8 octets)
  - 8 octets de plus
    - suffisant pour déclarer un chunk
- les données de la cible
  - en les déchiffrant à l’avance

“AngeCryption”

A partir de 2 fichiers, ***S***ource et ***C***ible,

- créer un fichier ***R***ésultat

R affiche

- S initialement
- C après chiffrement par AES-CBC(*clé, VI)*

le fichier R

R est fait de:

1. S
2. chunks de C, pré-déchiffrés.

R chiffré

une fois chiffré, R devient:

1. une signature PNG
2. un chunk couvrant les chunks de S chiffrés
3. les chunks de C

Etape par étape

Données initiales

On définit la clé, les fichiers S et C

clé AngeCryptionKey!

S 

C 

Vérifications

- S et C sont des PNGs
  - le PNG tolère les données ajoutées en fin de fichier
  - le PNG tolère l'ajout de chunk de stockage
    - dès le début du fichier, après la signature
- S tient dans un seul chunk de C
  - ne dépasse pas la taille maximale
- AES-128 a une taille de bloc de 16 octets
  - suffisant pour déclarer un chunk de stockage

Déterminer le 1er bloc chiffré

- R aura le même premier bloc que S
- une fois chiffré, R commence par:
  a. signature PNG de 8 octets
  b. un chunk de stockage
     - qui englobe tous les chunks de S
       1. S fait `14022` octets, donc `14016` octets de chunks
       2. `14016` sera encodé `000036c0`
     - notre chunk de stockage a comme type: **rmll**

       ⇒ ignoré

Premier bloc de R chiffré, C1:

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

Blocs en clair / chiffré

Premier bloc de R, P1:

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

Premier bloc de R chiffré, C1:

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

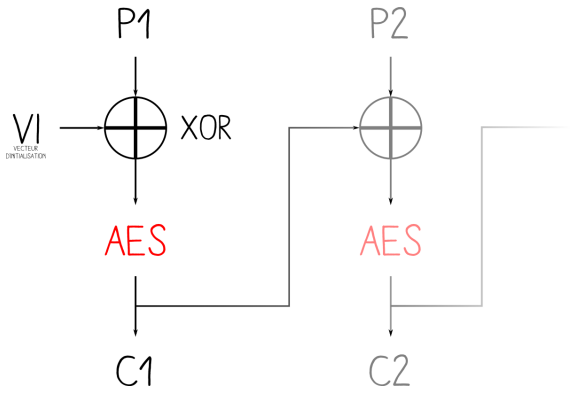
Déterminer le VI

On a les blocs P1 et C1, et la clé

1. Déchiffrer C1
2. XOR avec P1

On obtient le VI qui chiffrera P1 en C1

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



Générer R

le VI est déterminé.

- On bourre (*padding*) S aux 16 octets supérieurs
- On le chiffre en AES-CBC avec nos paramètres:

→ avec ce VI (après chiffrement)
S débutera par:

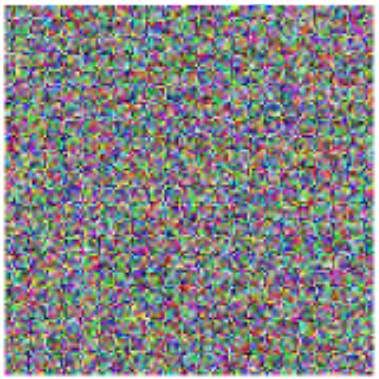
1. une signature
2. un chunk ‘rmll’ (couvrant le reste de S)

Ajuster le chunk de stockage

1. Les chunks finissent par un CRC32
   a. on le calcule à partir des informations chiffrées
2. On rajoute les chunks en clair de C
3. On déchiffre le fichier après bourrage

Résultat

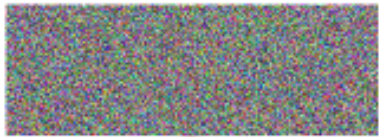
1. signature
2. chunks de S
3. bourrage
4. chunks de C (pré-déchiffrés)



```
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-
```

Résultat, chiffré

1. signature
2. chunk de stockage
   a. CRC32
3. chunks de C
4. bourrage



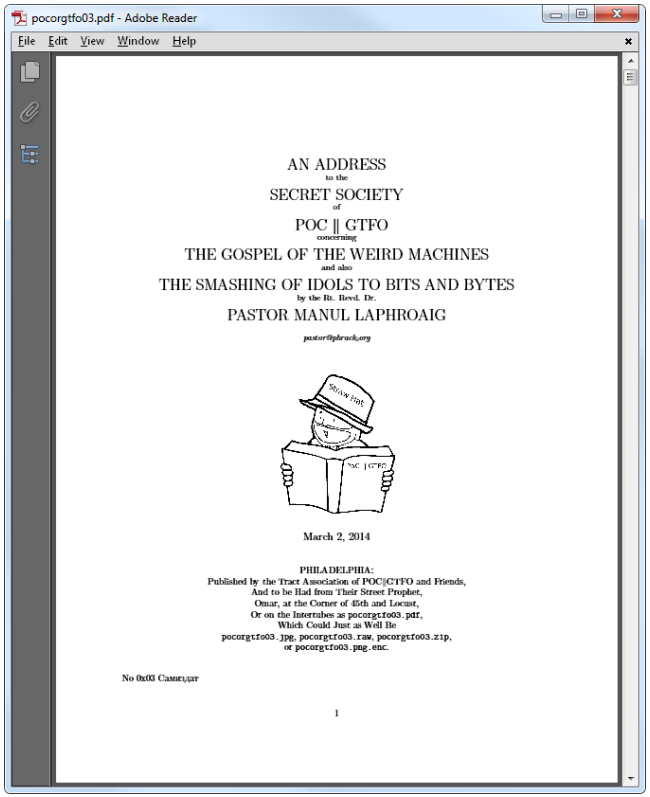
```
0000:  89 50 4E 47-0D 0A 1A 0A-00 00 36 C0-72 6D 6C 6C  ëPNG      6+rmll
0010:  9A 3E 30 1C-F1 D6 E1 41-B7 38 DB A1-5A 71 57 8F  Ü>0 ±+ßA+8¦íZqWÅ
0020:  6E 49 A0 D5-76 4C 33 7D-9B CA 44 B8-72 27 48 D9  nIá+vL3}¢-D+r'H+
0030:  64 20 A6 7F-38 D8 89 4A-9F 5F 92 45-17 5D 70 BA  d ª 8+ëJƒ_ÆE ]p¦
...
36A0:  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.............
```

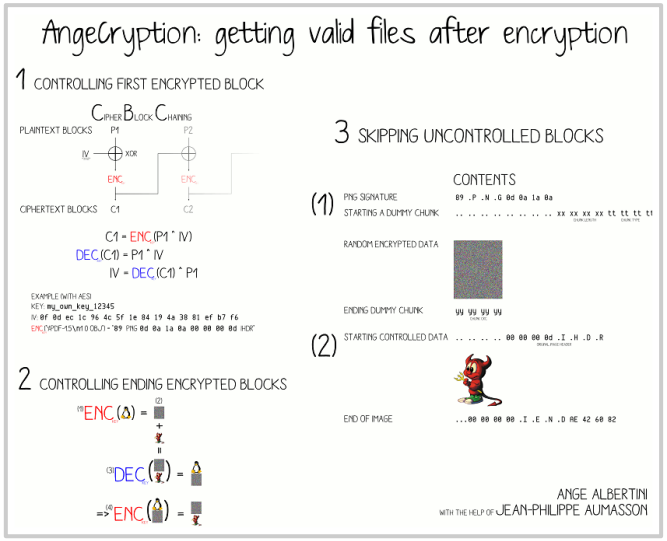
Rappels

En généralisant:

- Le format source tolère des données ajoutées
- Le format cible peut contenir une signature et une déclaration de chunk dans le premier bloc chiffré
- S tient dans un seul chunk (pas trop gros)

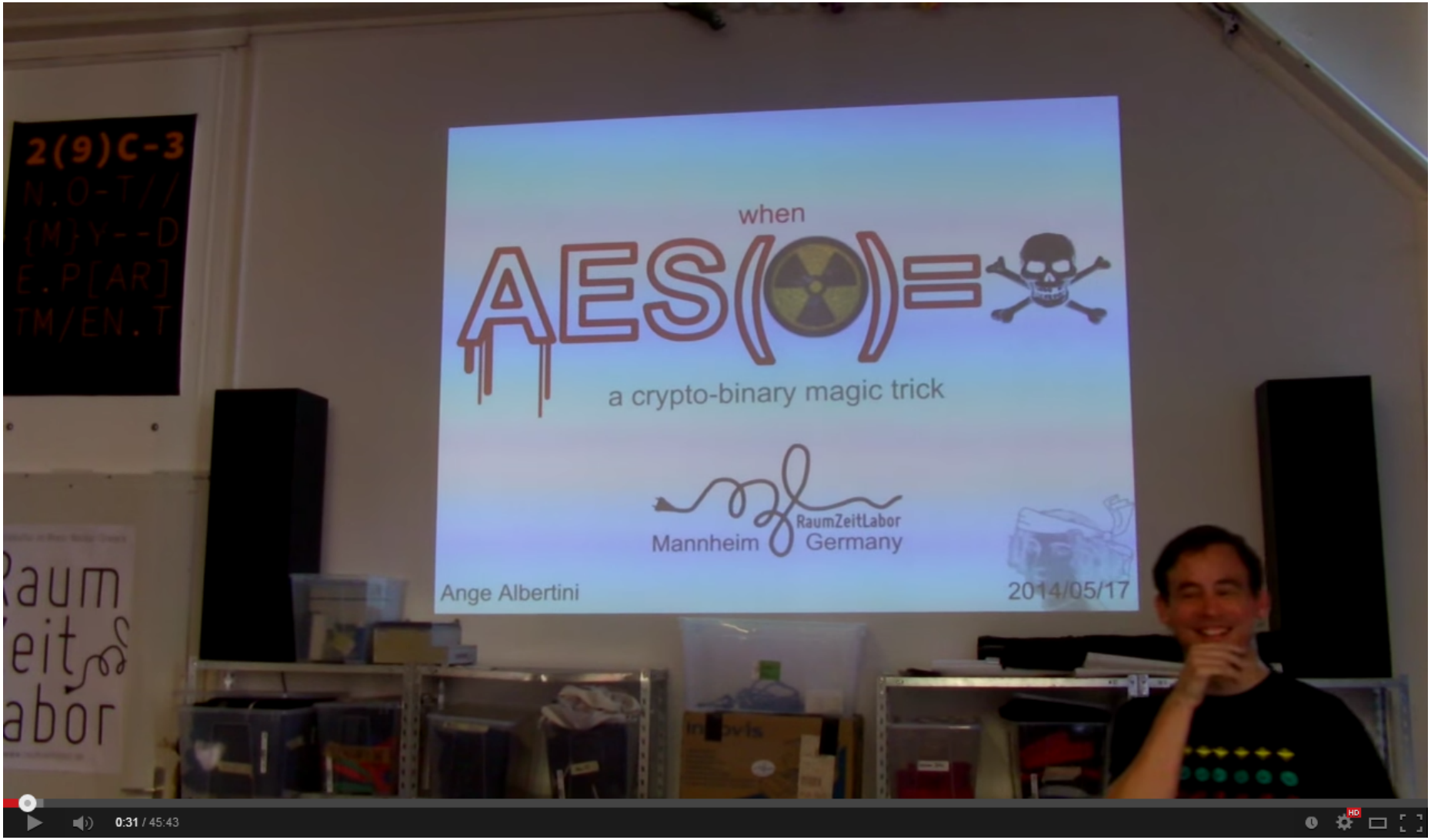
Utilisable avec d’autres algorithmes,
en chiffrant ou déchiffrant,
en combinant plusieurs formats à la fois





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 est un PDF qui se chiffre en un PNG
(et qui connait son propre VI de chiffrement)



Pour en savoir plus (en Anglais):
https://speakerdeck.com/ange/when-aes-equals-episode-v
https://www.youtube.com/watch?v=wbHkVZfCNuE

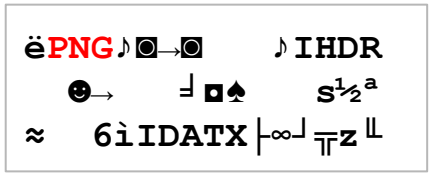
Jouons avec TrueCrypt

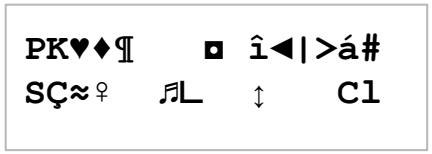
“Vrai Chiffrement”

L’outil TrueCrypt

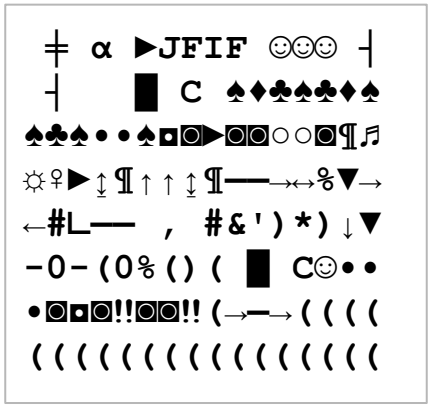
- Crée et utilise un volume de stockage virtuel
  - Chiffré
  - Transparent pour le système

Le volume est inutilisable sans le mot de passe.

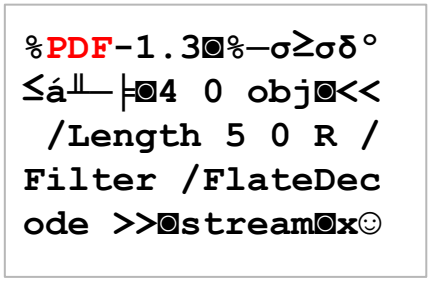


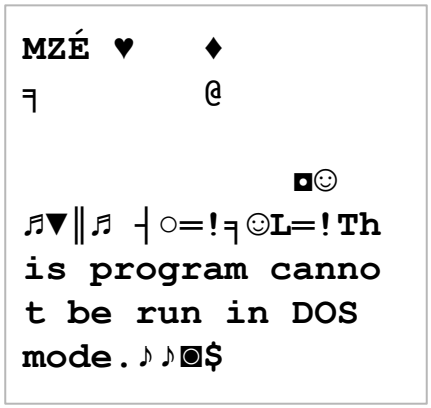


ZIP

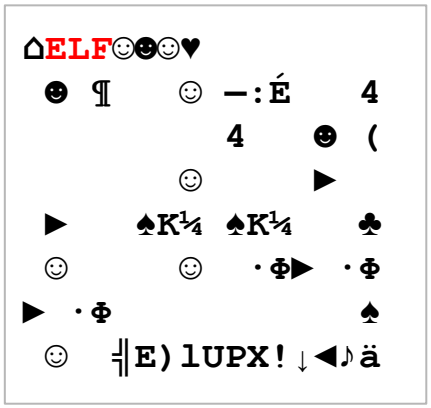


JPG

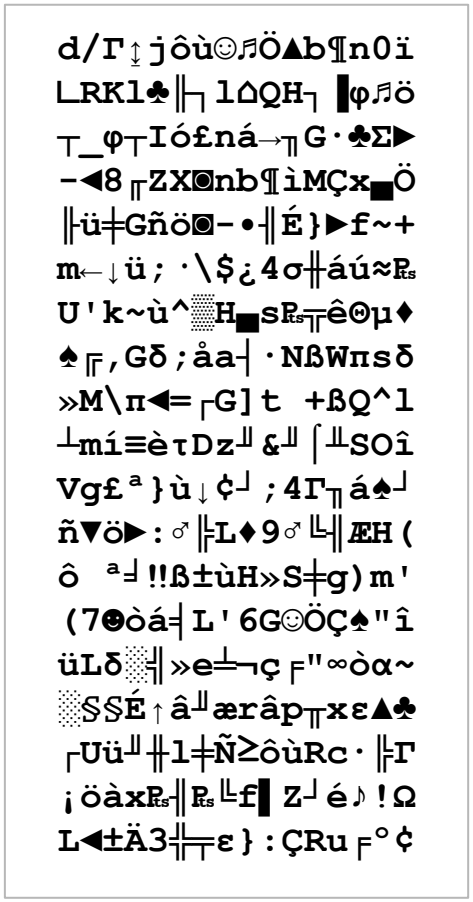


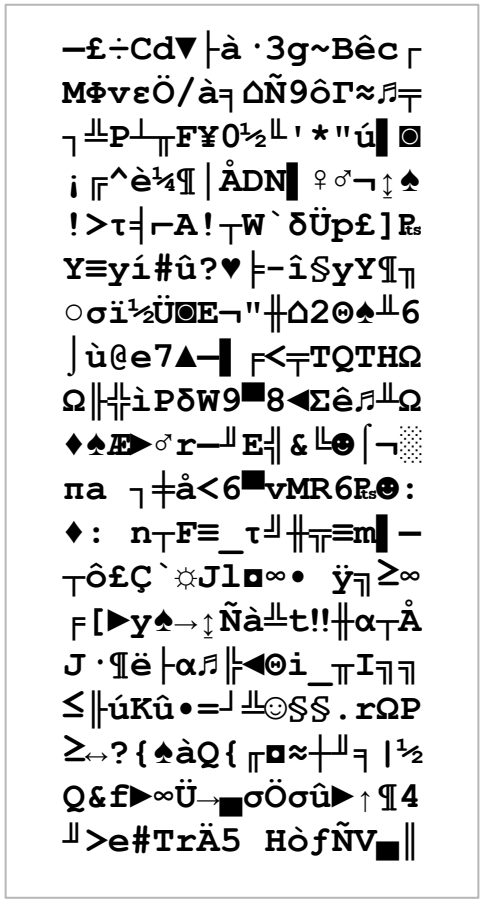


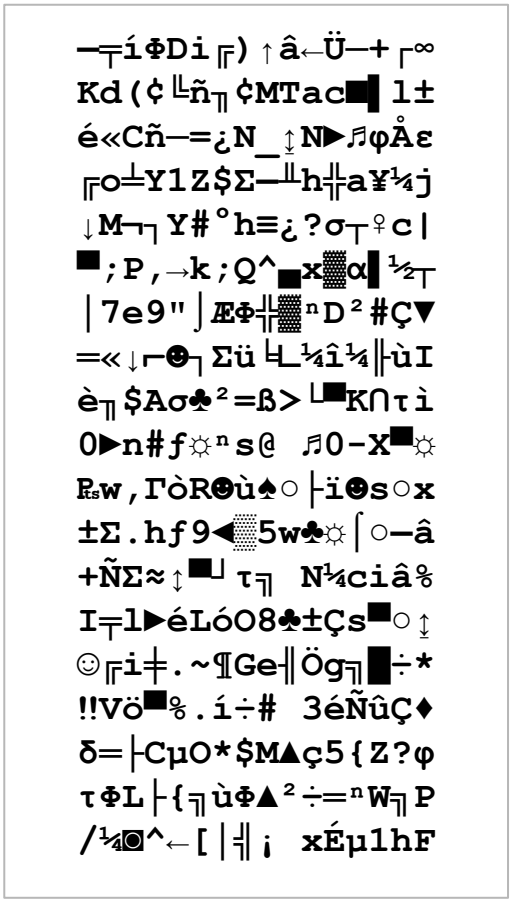
PE



En-têtes de différents formats de fichiers standards.





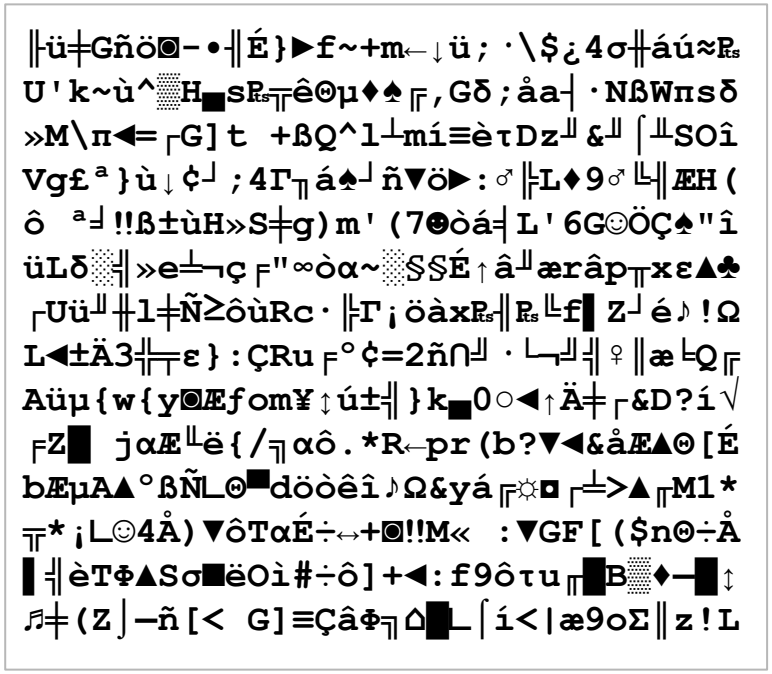


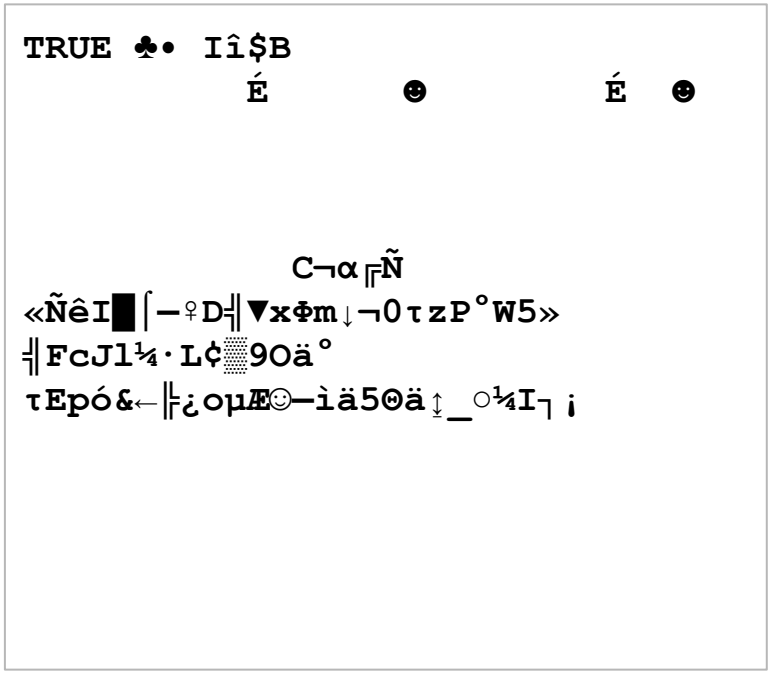
En-têtes de fichiers TrueCrypt.

Un contenu aléatoire ?

Un format fait pour ne PAS être identifié

- sauf si vous avez le mot de passe
- apparence aléatoire
  - vous pouvez nier que c'est un volume TrueCrypt
- il y ***a*** un en-tête
  - mais il est chiffré





En-tête TrueCrypt avant/après déchiffrement

Combien avez-vous de fichiers 100% aléatoires?

pas si discret

Détection de volumes *potentiels*

- aucun en-tête de fichier connu
- “grosse taille”, en général
- taille multiple de 512
- contenu aléatoire
  - très haute *entropie*

Juste un mot de passe ?

Si le chiffrement dépend juste du mot de passe,

TrueCrypt est vulnérable aux attaques par tables *rainbow*.

Sel

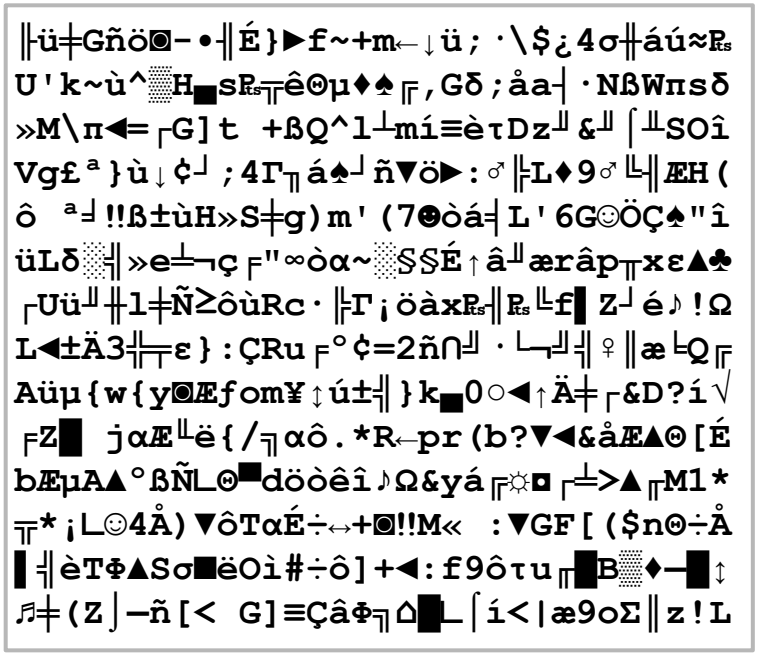
Le fichier débute par 64 octets de sel

- contenu aléatoire
- combiné au mot de passe
- utilisé pour déchiffrer l'en-tête

⇒ pré-calcul impossible

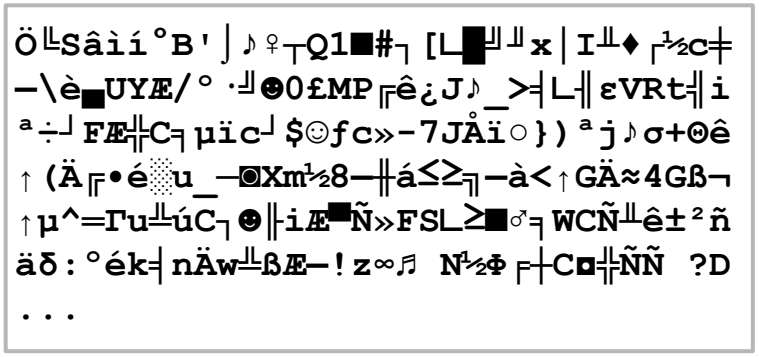
⇒ rainbow tables inefficaces

Sel
(pour déchiffrer l’en-tête)



En-tête

- chiffré avec le sel et le mot de passe
- contient les clés pour déchiffrer le contenu du volume



Contenu du volume

- chiffré avec les clés contenues dans l’en-tête

Structure de volume TrueCrypt

Si on modifie le sel, il suffit de re-chiffrer l'en-tête

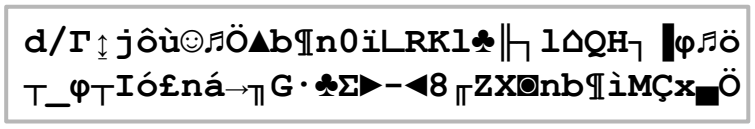
pas besoin de toucher le contenu du volume
(les clés de volume n'ont pas changé)

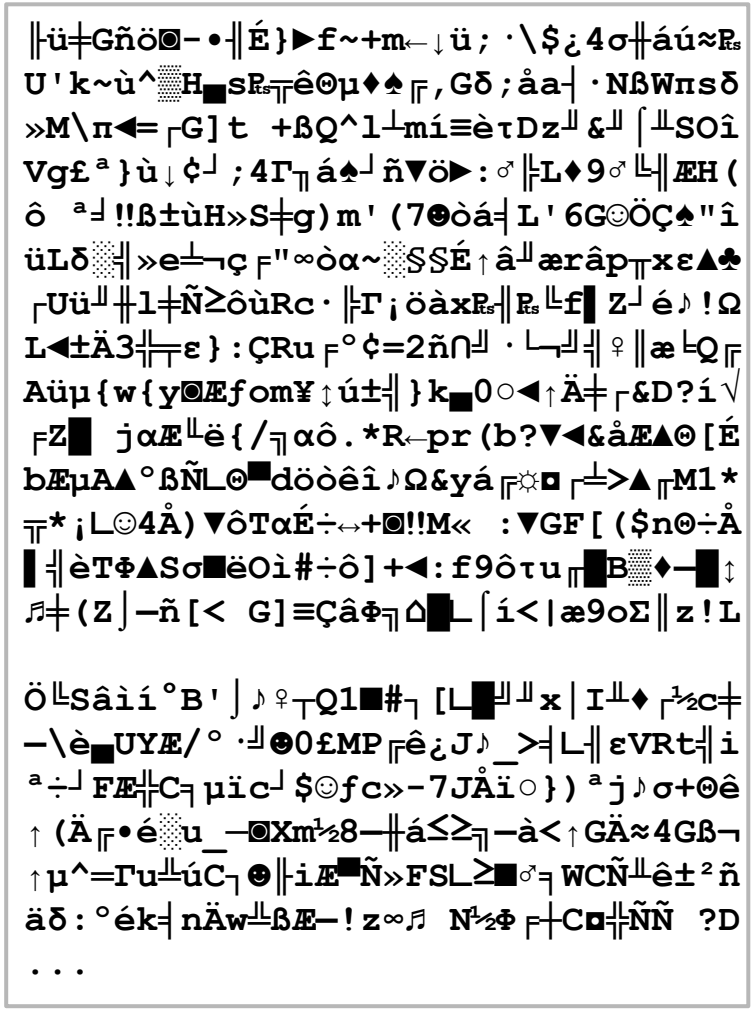
Intégrer un volume TrueCrypt dans un autre fichier

- c’est plus discret
- les 2 formats restent valides

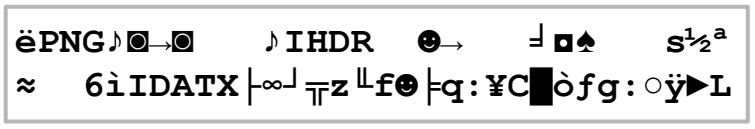
Stratégie

1. Modifier l'hôte pour laisser de l'espace au début
   - créer un chunk 'inutile' de la taille du volume
2. Copier l'en-tête et le contenu du volume
   - l'en-tête en clair n'est pas modifié,
     donc le contenu du volume non plus
3. Déchiffrer l'en-tête du volume
   - avec le sel initial
4. Re-chiffrer l'en-tête
   - avec le nouveau sel du début de l'hôte

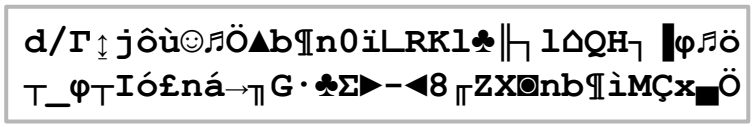


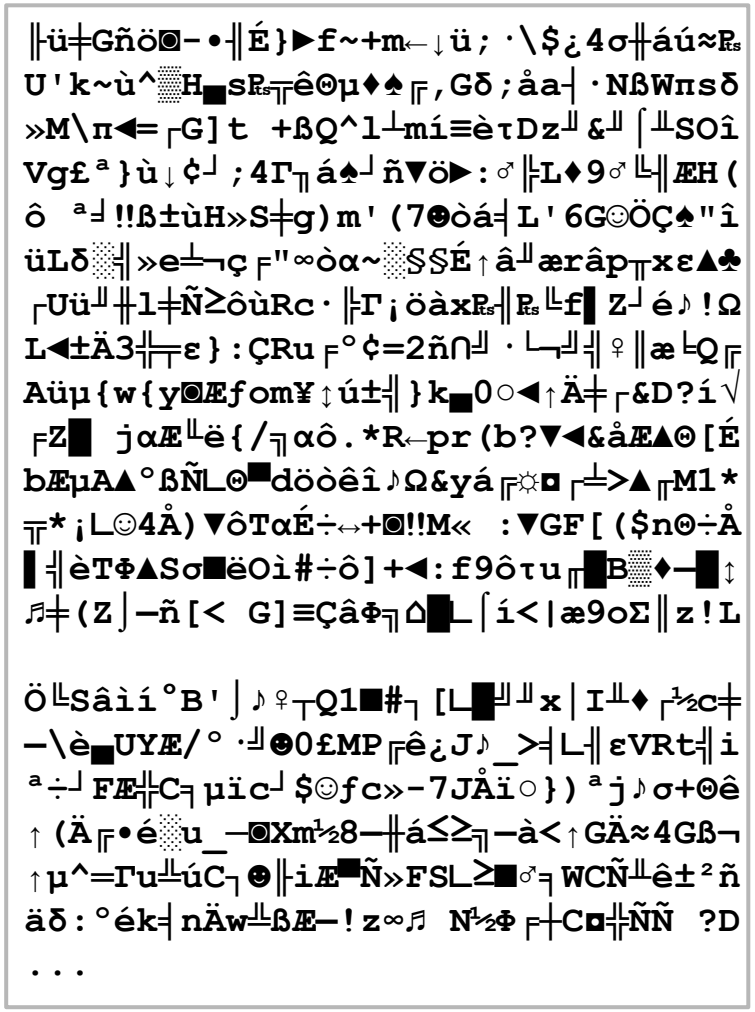


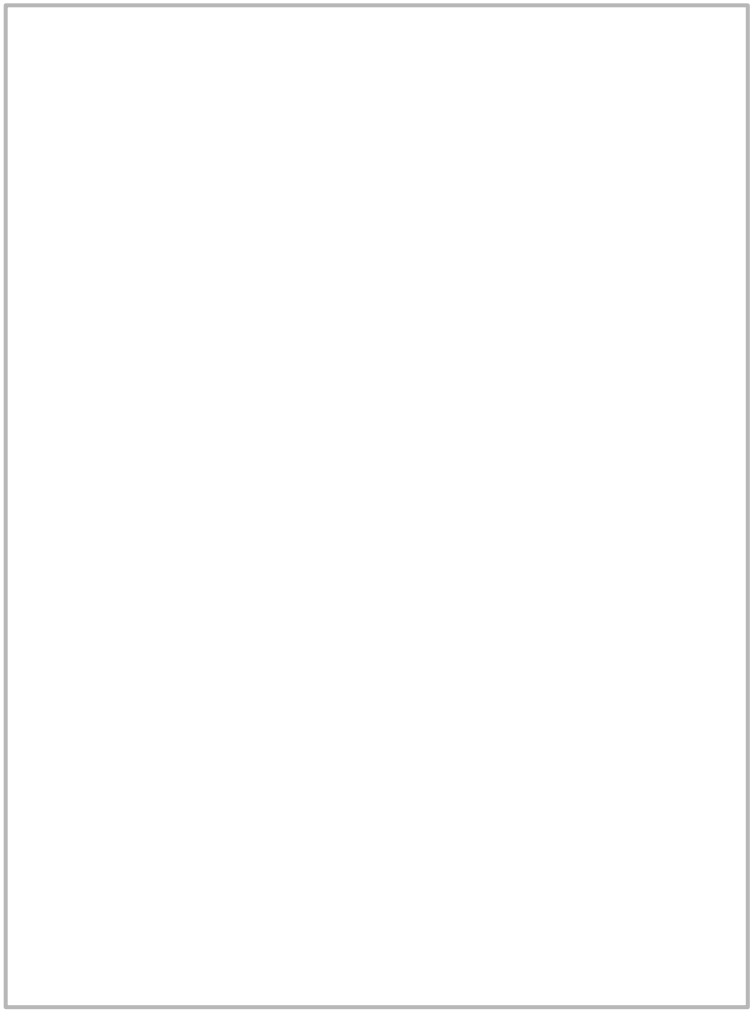
Volume TrueCrypt



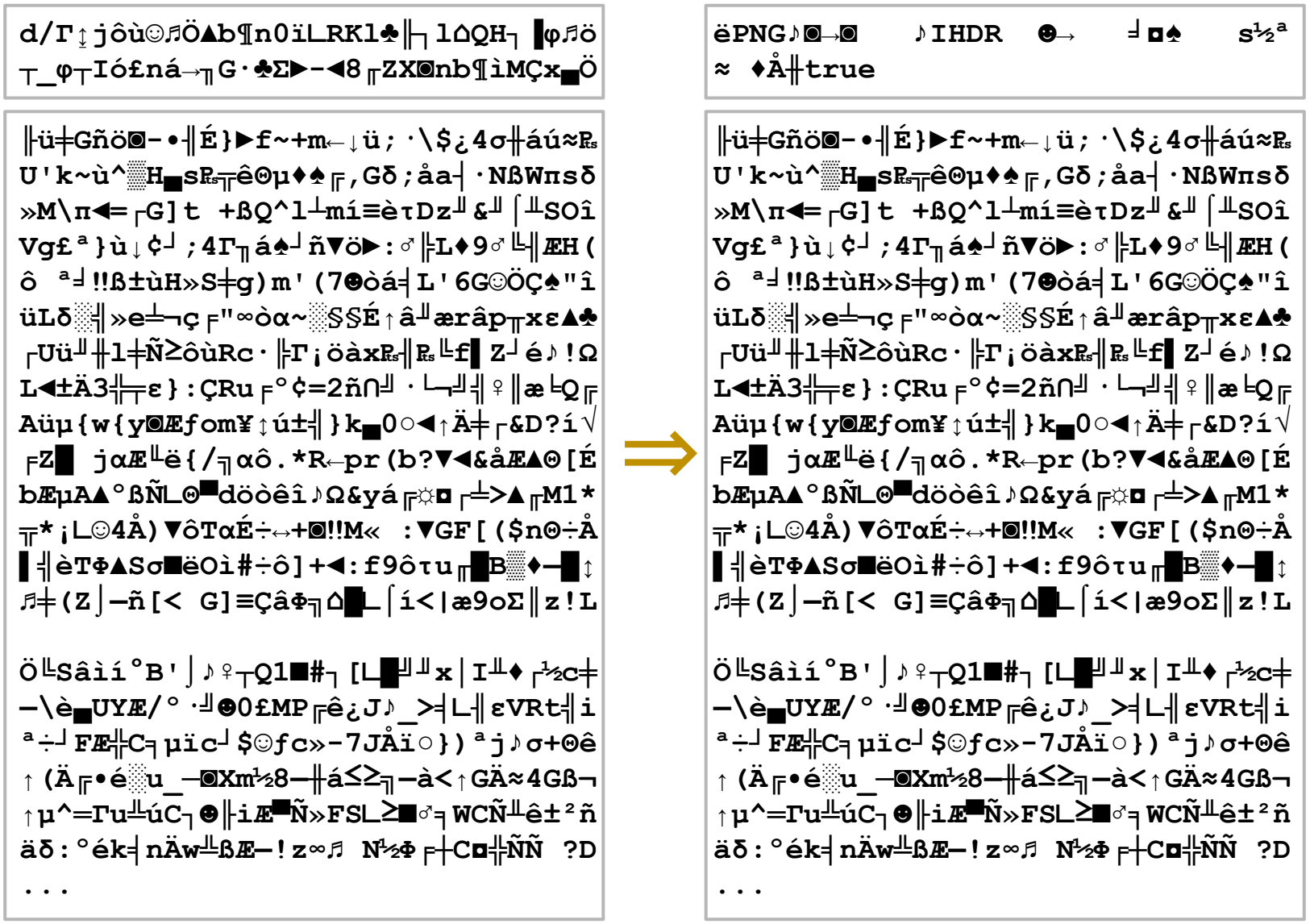
Image



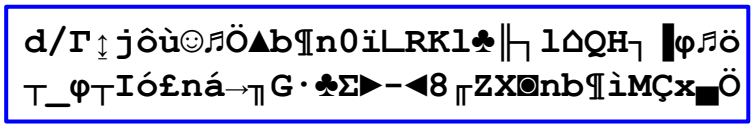


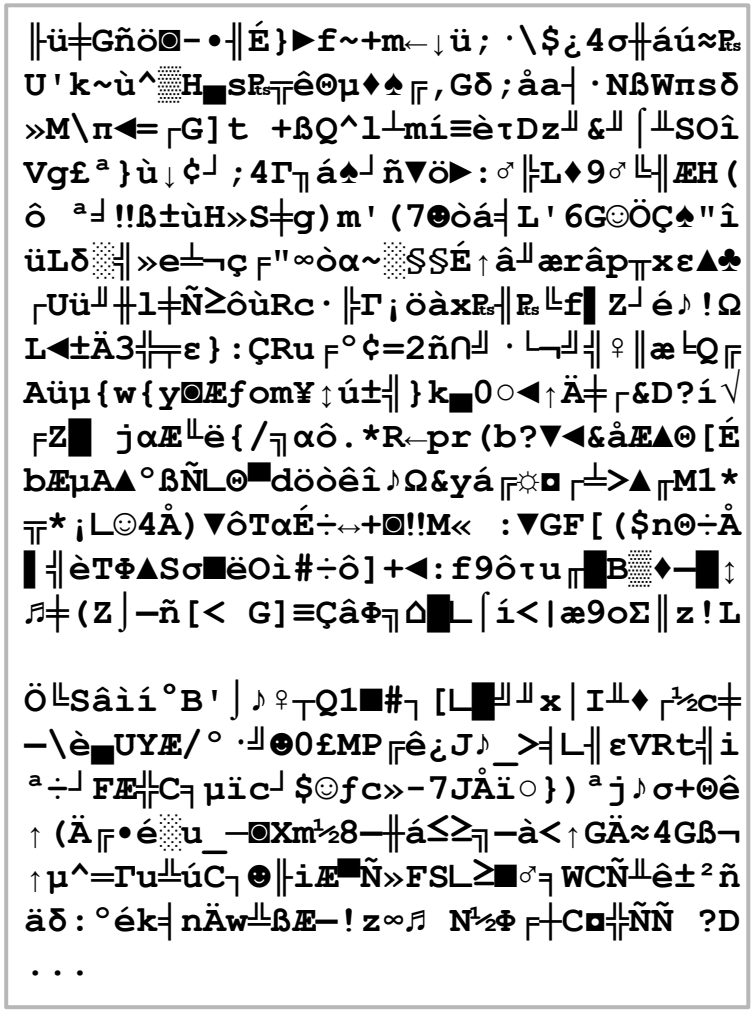


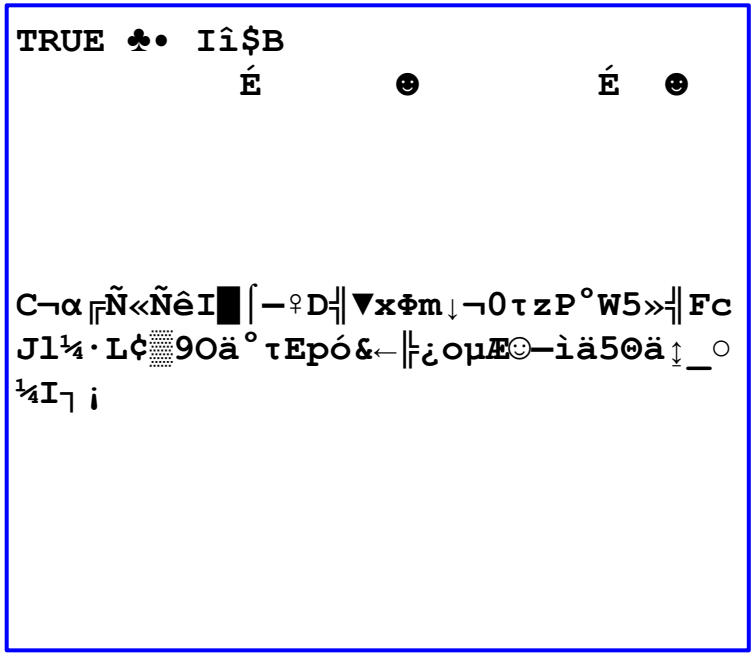
Création d’espace dans le fichier hôte pour recevoir le volume.

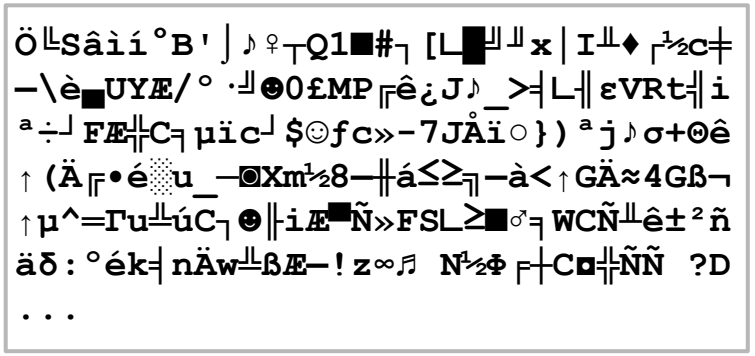


Copie du volume dans l’espace créé.

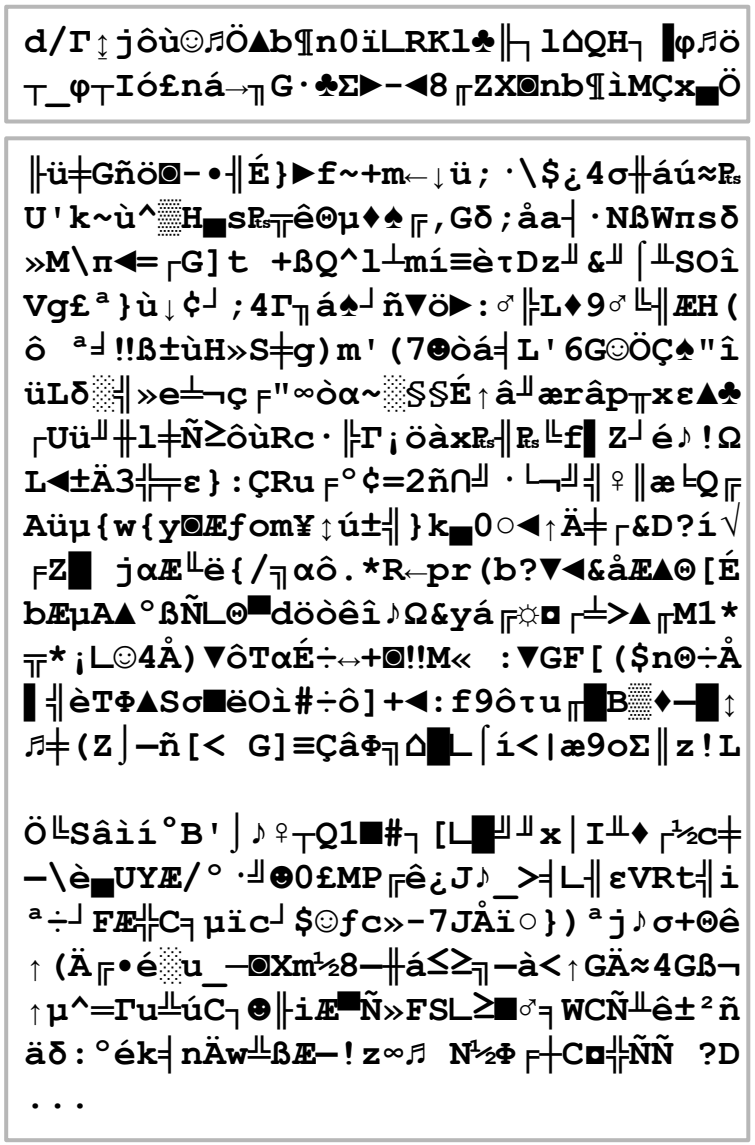


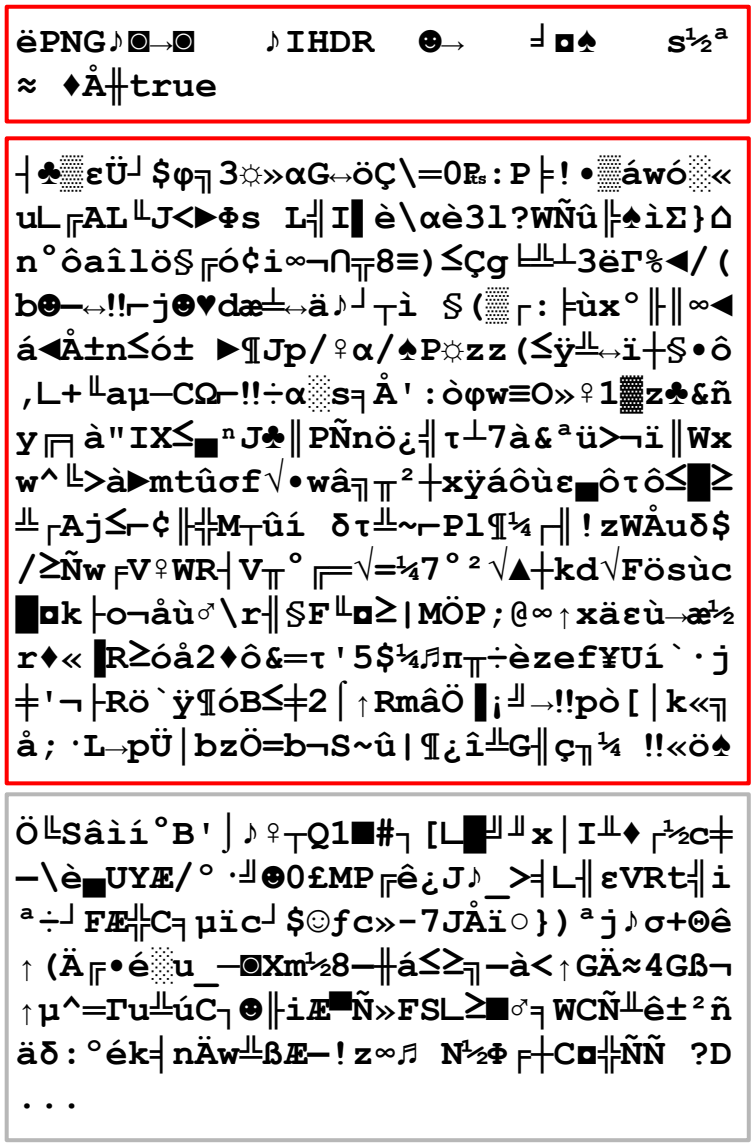






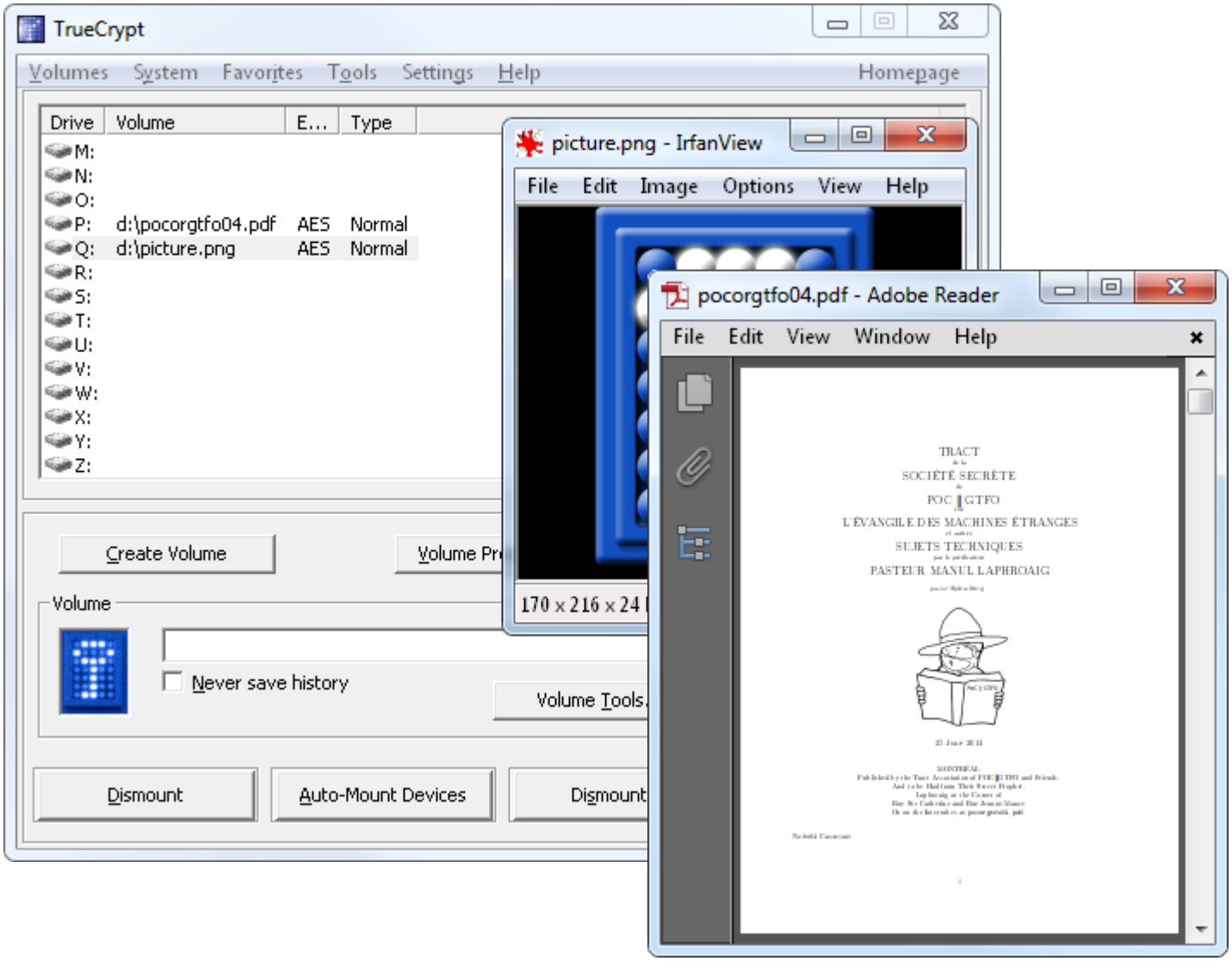
Déchiffrement de l’en-tête avec le sel du volume.





Re-chiffrement de l’en-tête avec le “sel” de l’hôte.

TrueCrypt	PNG	
Sel	Signature + en-tête Déclaration de chunk	ëPNG♪◙→◙ ♪IHDR ☻→ ⁴◘♠ s½ª ≈ ♦Å╫true
En-tête	Donnés du chunk	┤♣▒εÜ┘$φ╕3☼»αG↔öÇ\=0₧:P╞!•▒áwó▒« uL╓AL╨J<►Φs L╣I▌è\αè3l?WÑû╠♠ìΣ}⌂ n°ôaîlö§╓ó¢i∞¬∩╥8≡)≤Çg╙╨┴3ëΓ%◄/(b☻─↔!!⌐j☻♥dæ┻↔ä♪┘┬ì §(▒┌:╞ùx°╟╢∞◄ á◄Å±n≤ó± ►¶Jp/♀α/♠P☼zz(≤ÿ╨↔ï┼§•ô ,L+╨aμ─CΩ─!!÷α▒s╕Å':òφw≡O»♀1▓z♣&ñ y╒╕à"IX≤▄ⁿJ♣╢PÑnö¿╣τ┴7à&ªü>¬ï╢Wx w^╚>à►mtûσf√•wâ╕╥²┼xÿáôùε▄ôτô≤▌≥ ╨┌Aj≤⌐¢╟╬M┬ûí δτ╨~⌐Pl¶¼┌╢!zWÅuδ$ /≥Ñw╒V♀WR┤V╥° ╒═√═¼7°²√▲┼kd√Fösùc ▌◘k├o¬åù♂\r╢§F╚◘≥|MÖP;@∞↑xäεù→æ½ r♦« ▌R≥óå2♦ô&=τ'5$¼♫π╥÷èzef¥Uí`·j ╪'¬├Rö`ÿ¶óB≤╪2⌠↑RmâÖ▌¡╝→!!pò[│k«╕ å;·L→pÜ│bzÖ=b¬S~û|¶¿î╨G╢ç╖¼ !!«ö♠
Contenu du volume		Ö╚Sâìí°B'⌡♪♀┬Q1■#┐[L▐╝╜x│I╨♦┌½c╪ ─\è▄UYÆ/° ·╝☻0£MP╒ê¿J♪_>╡L╢εVRt╣i ª÷┘FÆ╬C╕μïc┘$☺ƒc»-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 ...
Fin du volume ✓	Fin du chunk ✓ Chunks originaux	



Des volumes TrueCrypt dans des fichiers standards
(toujours utilisables normalement)

Conclusion 1/2

- On peut insérer des données externes dans un fichier binaire standard
- Ces informations peuvent contenir
  - un autre fichier standard, après [dé]chiffrement
  - un volume TrueCrypt

Conclusion 2/2

- Pas besoin de tout comprendre avec la crypto
- Mieux vaut y aller pas à pas
  - demander à un expert ;)
  - pas facile à déboguer

- Chiffré ne rime pas forcément avec aléatoire

- exemples: http://bit.ly/1n63yKP
  (http://corkami.googlecode.com/svn/trunk/src/angecryption/rmll)

Remerciements

@veorq @doegox

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

@angealbertini

corkami.com

```
D:\test>advpng -z -4 logo11w.png
        14022        12599  89% logo11w.png
        14022        12599  89%
```

“je peux faire mieux que Google!”

Réduire la taille du logo Google (avec leur propre algorithme).