The Advanced Encryption Standard

Introduction

The Advanced Encryption Standard (AES) was adopted in 2001 by the U.S. National Institute for Standards and Technology (NIST) - to replace the less secure Data Encryption Standard (DES) adopted in 1976 - and has found worldwide acceptance. AES is a block cipher achieving both excellent resistance against most known attacks and fast encryption/decryption. It is recognized as the de facto standard for modern symmetric key cryptosystems and is therefore widely used in industrial applications. AES plays a critical role in secure data transmission protocols over the Internet - e.g. Secure Sockets Layer (SSL), Transport Layer Security (TLS), Wi-Fi protected access (WPA2), Internet phone (Skype), secure shell network (SSH), virtual private network (VPN). AES has also found applications in antivirus software and protected databases (e.g. website passwords, credit card numbers, PIN-codes etc).

The purpose of this practical work using python is threefold :

1. constructing $\mathbb{F}_{2^8}$ and using arithmetic in this Galois field
2. programming all AES transformations based on 8-bit bytes processing, expressed as operations in $\mathbb{F}_{2^8}$
3. end-to-end programming of the AES128 encryption/decryption standard (i.e. the key and the block length is 128 bits).

I. $\mathbb{F}_{2^8}$ Galois field construction and arithmetic

Consider the Galois field $\mathbb{F}_2 = (\{0,1\},+,\times)$, where $+$ and $\times$ denote modulo-2 addition and multiplication. Let $\mathbb{F}_2[x]$ be the set of polynomials in the variable x, with coefficients in $\mathbb{F}_2$. Considering the degree-8 irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ in $\mathbb{F}_2[x]$, we refer to the Galois field $\mathbb{F}_{2^8}$ as $\mathbb{F}_2[x]/m(x)$ which is by definition the set of residual classes modulo $m(x)$. Thus any element in $\mathbb{F}_{2^8}$ admits a polynomial representation of the form $b(x) = b_7x^7 + b_6x^6 + \cdots + b_0$, or equivalently a binary representation of the form $(b_7, b_6, \ldots, b_0)$ (which can also be converted to a more compact hexadecimal form - see Fig. 6-7).

The proposed implementation can be found in the module `my_gf_functions.py` along with the test script `testGaloisField.py` available in the folder /GaloisField. The questions below will help you to complete the missing parts indicated by """FILL IN MISSING CODE""".

– Addition in $\mathbb{F}_{2^8}$: Let $a(x) = a_7x^7 + a_6x^6 + \cdots + a_0$ and $b(x) = b_7x^7 + b_6x^6 + \cdots + b_0$ be the polynomial form of two elements in $\mathbb{F}_{2^8}$, it follows that

$$c(x) = a(x) + b(x) \mod m(x) = c_7x^7 + c_6x^6 + \cdots + c_0 \text{ in } \mathbb{F}_{2^8}, \tag{1}$$

where $(c_7, c_6, \ldots, c_0) = (a_7, a_6, \ldots, a_0) \bigoplus (b_7, b_6, \ldots, b_0)$ and $\bigoplus$ denotes the bitwise xor operator since $\deg(c(x)) < 8$.

Note that the function `add(.,.)` implements addition in $\mathbb{F}_{2^8}$.

– Multiplication by x in $\mathbb{F}_{2^8}$: Since $xb(x) = b_7x^8 + b_6x^7 + \cdots + b_0x$, noting that $x^8 - m(x) = x^4 + x^3 + x + 1 = [1B]$ we have

$$xb(x) \mod m(x) \text{ in } \mathbb{F}_{2^8} = \begin{cases} b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \text{ if } b_7 = 0 \\ b_6x^7 + b_5x^6 + b_4x^5 + (b_3+1)x^4 + (b_2+1)x^3 + b_1x^2 + (b_0+1)x + 1 \text{ if } b_7 = 1. \end{cases} \tag{2}$$

Note that the function `xtime(.)` implements multiplication by x in $\mathbb{F}_{2^8}$. As a byproduct, the function `mult_xtime(.,.)` is a naive implementation for $c(x) = a(x)b(x) \mod m(x)$ in $\mathbb{F}_{2^8}$ using a recursive application of `xtime(.)`.

I.1. Write a script to check the `xy` $\mapsto$ `xtime(xy)` table in Fig. 1.

I.2. Compute $[A1] + [12]$ in $\mathbb{F}_{2^8}$. Write the corresponding code using `add(.,.)`.

I.3. Compute $[57].[83]$ in $\mathbb{F}_{2^8}$. Write the corresponding code using `mul_xtime(.,.)`.

I.4. Write a script to check that $g = [03]$ generates $\mathbb{F}_{2^8}$.

I.5. Let Logtable[.] be the tabular representation of $g^i \mapsto i$ for $i = 0, \ldots, 254$ (use the convention Logtable[0]=0). Conversely, let Alogtable[.] be the tabular representation of $i \mapsto g^i$ for $i = 0, \ldots, 255$. Complete the function definition `def Generate_Logtable_Alogtable()` : returning Logtable[.] and Alogtable[.]. The final result can be double checked with the tables in Fig. 3 and 4.

I.6. Noting that multiplication in $\mathbb{F}_{2^8}$ using the naive implementation `mult_xtime(.,.)` might be a threat wrt side-channel attacks monitoring power consumption or timing, propose an alternative implementation using the generator g in `def mult(a,b)`:. Check its validity on an example by recomputing $[57].[83]$ in $\mathbb{F}_{2^8}$.

I.7. Using the generator g, complete the definition of the function `def inv(a)` : that maps $a \mapsto a^{-1}$ in $\mathbb{F}_{2^8}$ ($0^{-1} = 0$ by convention in AES) - see Fig. 2 for a tabular representation. Check its validity on an example by computing $[83]^{-1}$ in $\mathbb{F}_{2^8}$.

I.8. Again using the generator g, complete the definition of the function `def div(a,b)` : that maps $(a,b) \mapsto a/b$ in $\mathbb{F}_{2^8}$. Check its validity on an example by computing $[01]/[83]$ in $\mathbb{F}_{2^8}$.

II. AES transformations

The proposed implementation of AES transformations can be found in the module `my_transformations_functions.py` along with the test script `testTransformations.py` available in the folder /Transformations. The questions below will help you to complete the missing parts indicated by """FILL IN MISSING CODE""".

II.1. Using the function `Sbox(.)` (resp. `InvSbox(.)`), write a script giving a tabular representation of $a \mapsto Sbox(a)$ (resp. $a \mapsto Sbox^{-1}(a)$) for all $a \in \mathbb{F}_{2^8}$. The final result can be double checked with the tables in Fig. 5.

II.2. Check their validity of the function `ShiftRows(.)` (resp. `InvShiftRows(.)`) implementing the Shift Rows transformation (resp. the inverse Shift Rows transformation), on the following 4×4

state matrix of elements in $\mathbb{F}_{2^8}$:

$$\begin{bmatrix} [87] & [F2] & [4D] & [97] \\ [6E] & [4F] & [90] & [EC] \\ [46] & [E7] & [4A] & [95] \\ [A6] & [8C] & [D8] & [95] \end{bmatrix}. \tag{3}$$

II.3. Complete the function definition `def MixColumn()` : implementing the Mix Column tranformation (resp. `def InvMixColumn()` : implementing the inverse Mix Column tranformation). Check their validity on the following 4×4 state matrix of elements in $\mathbb{F}_{2^8}$:

$$\begin{bmatrix} [0E] & [0B] & [0D] & [09] \\ [09] & [0E] & [0B] & [0D] \\ [0D] & [09] & [0E] & [0B] \\ [0B] & [0D] & [09] & [0E] \end{bmatrix}. \tag{4}$$

II.4. The function `ExpandKey(,)` implements the key expansion algorithm (the first argument is the key considered as a length-16 vector of bytes in integer form, while the second argument in the number of rounds). AES128 using 10 rounds, apply the key expansion algorithm to the following key vector in hex form

`original_key1=[[0F],[15],[71],[C9],[47],[D9],[E8],[59],[0C],[B7],[AD],[D6],[AF],[7F],[67],[98]]`.

By flipping the 8-th key bit in `original_key1`, observe the evolution of the number of different key bits at each round.

III. AES128 Encryption/Decryption

The proposed implementation of AES Encryption/Decryption can be found in the module `my_aes128_functions.py` available in the folder /AES128. The questions below will help you to complete the missing parts indicated by """FILL IN MISSING CODE""".

III.1. Complete the function definition `def Encrypt( , , ):` (resp. `def Decrypt( , , ):`) implementing AES128 ciphering (resp. deciphering) on a single block.

III.2. Use the script `testAES128.py` to test the correctness of AES128 Encryption/Decryption with 10 rounds, over a single block with plaintext vector in hex form chosen as :

`original_message1=[[01],[23],[45],[67],[89],[AB],[CD],[EF],[FE],[DC],[BA],[98],[76],[54],[32],[10]]`

and key vector in hex form chosen as :

`original_key1=[[0F],[15],[71],[C9],[47],[D9],[E8],[59],[0C],[B7],[AD],[D6],[AF],[7F],[67],[98]]`.

III.3. By flipping the 8-th key bit in `original_message1` (resp. `original_key1`), observe the evolution of the number of different key bits at each round. Does the AES128 standard with 10 rounds have good confusion properties ?

III.4. Reproduce the experiment of question III.3. with more than 10 rounds. Does that improve

significantly the AES128 standard's confusion properties ?

III.5. When ciphering/deciphering a text, the corresponding ASCII character string must be converted to plaintexts of 128 bits and padded with spaces so as to get an integer number of plaintexts (see the conversion table in Fig. 8). Use the script `CipherDecipherText.py` for the sake of AES128 ciphering/deciphering of the following sentence from Cicero's book "De finibus bonorum et malorum" written in 45 BC :

```
"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua."
```

By adding more text, plot the average CPU execution time of AES1328 encryption and decryption as a function of the text length. Is the trend linear, quadratic, exponential ?

Bibliography

J. Daemen and V. Rijmen, "The Design of Rijndael : AES - The Advanced Encryption Standard", Springer, 2002.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	00	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
	1	20	22	24	26	28	2A	2C	2E	30	32	34	36	38	3A	3C	3E
	2	40	42	44	46	48	4A	4C	4E	50	52	54	56	58	5A	5C	5E
	3	60	62	64	66	68	6A	6C	6E	70	72	74	76	78	7A	7C	7E
	4	80	82	84	86	88	8A	8C	8E	90	92	94	96	98	9A	9C	9E
	5	A0	A2	A4	A6	A8	AA	AC	AE	B0	B2	B4	B6	B8	BA	BC	BE
	6	C0	C2	C4	C6	C8	CA	CC	CE	D0	D2	D4	D6	D8	DA	DC	DE
	7	E0	E2	E4	E6	E8	EA	EC	EE	F0	F2	F4	F6	F8	FA	FC	FE
	8	1B	19	1F	1D	13	11	17	15	0B	09	0F	0D	03	01	07	05
	9	3B	39	3F	3D	33	31	37	35	2B	29	2F	2D	23	21	27	25
	A	5B	59	5F	5D	53	51	57	55	4B	49	4F	4D	43	41	47	45
	B	7B	79	7F	7D	73	71	77	75	6B	69	6F	6D	63	61	67	65
	C	9B	99	9F	9D	93	91	97	95	8B	89	8F	8D	83	81	87	85
	D	BB	B9	BF	BD	B3	B1	B7	B5	AB	A9	AF	AD	A3	A1	A7	A5
	E	DB	D9	DF	DD	D3	D1	D7	D5	CB	C9	CF	CD	C3	C1	C7	C5
	F	FB	F9	FF	FD	F3	F1	F7	F5	EB	E9	EF	ED	E3	E1	E7	E5

FIGURE 1 – Tabular representation of `xy` $\mapsto$ `xtime(xy)` in $\mathbb{F}_{2^8}$ in hexadecimal form.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
	1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
	2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
	3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
	4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
	5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
	6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
	7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
	8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
	9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
	A	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
	B	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
	C	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
	D	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
	E	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
	F	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

FIGURE 2 – Tabular representation of `xy` $\mapsto$ `(xy)`$^{-1}$ in $\mathbb{F}_{2^8}$ in hexadecimal form.

```
word8 Logtable[256] = {
  0,  0, 25,  1, 50,  2, 26,198, 75,199, 27,104, 51,238,223,  3,
100,  4,224, 14, 52,141,129,239, 76,113,  8,200,248,105, 28,193,
125,194, 29,181,249,185, 39,106, 77,228,166,114,154,201,  9,120,
101, 47,138,  5, 33, 15,225, 36, 18,240,130, 69, 53,147,218,142,
150,143,219,189, 54,208,206,148, 19, 92,210,241, 64, 70,131, 56,
102,221,253, 48,191,  6,139, 98,179, 37,226,152, 34,136,145, 16,
126,110, 72,195,163,182, 30, 66, 58,107, 40, 84,250,133, 61,186,
 43,121, 10, 21,155,159, 94,202, 78,212,172,229,243,115,167, 87,
175, 88,168, 80,244,234,214,116, 79,174,233,213,231,230,173,232,
 44,215,117,122,235, 22, 11,245, 89,203, 95,176,156,169, 81,160,
127, 12,246,111, 23,196, 73,236,216, 67, 31, 45,164,118,123,183,
204,187, 62, 90,251, 96,177,134, 59, 82,161,108,170, 85, 41,157,
151,178,135,144, 97,190,220,252,188,149,207,205, 55, 63, 91,209,
 83, 57,132, 60, 65,162,109, 71, 20, 42,158, 93, 86,242,211,171,
 68, 17,146,217, 35, 32, 46,137,180,124,184, 38,119,153,227,165,
103, 74,237,222,197, 49,254, 24, 13, 99,140,128,192,247,112,  7};
```

FIGURE 3 – Tabular representation of $g^i \mapsto i$ in $\mathbb{F}_{2^8}$ for $i = 0,\ldots,254$, where $g = [03]$ (g^i is the index of the array in dec form while i is the corresponding value - use the convention Logtable[0]=0).

```
word8 Alogtable[256] = {
  1,  3,  5, 15, 17, 51, 85,255, 26, 46,114,150,161,248, 19, 53,
 95,225, 56, 72,216,115,149,164,247,  2,  6, 10, 30, 34,102,170,
229, 52, 92,228, 55, 89,235, 38,106,190,217,112,144,171,230, 49,
 83,245,  4, 12, 20, 60, 68,204, 79,209,104,184,211,110,178,205,
 76,212,103,169,224, 59, 77,215, 98,166,241,  8, 24, 40,120,136,
131,158,185,208,107,189,220,127,129,152,179,206, 73,219,118,154,
181,196, 87,249, 16, 48, 80,240, 11, 29, 39,105,187,214, 97,163,
254, 25, 43,125,135,146,173,236, 47,113,147,174,233, 32, 96,160,
251, 22, 58, 78,210,109,183,194, 93,231, 50, 86,250, 21, 63, 65,
195, 94,226, 61, 71,201, 64,192, 91,237, 44,116,156,191,218,117,
159,186,213,100,172,239, 42,126,130,157,188,223,122,142,137,128,
155,182,193, 88,232, 35,101,175,234, 37,111,177,200, 67,197, 84,
252, 31, 33, 99,165,244,  7,  9, 27, 45,119,153,176,203, 70,202,
 69,207, 74,222,121,139,134,145,168,227, 62, 66,198, 81,243, 14,
 18, 54, 90,238, 41,123,141,140,143,138,133,148,167,242, 13, 23,
 57, 75,221,124,132,151,162,253, 28, 36,108,180,199, 82,246,  1};
```

FIGURE 4 – Tabular representation of $i \mapsto g^i$ in $\mathbb{F}_{2^8}$ for $i = 0,\ldots,255$, where $g = [03]$ (i is the index of the array while g^i is the corresponding value in dec form).

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b) Inverse S-box

FIGURE 5 – Tabular representation of the AES Sbox and Sbox inverse in hexadecimal form.

DEC	HEX	BIN	DEC	HEX	BIN	DEC	HEX	BIN
0	00	00000000	43	2B	00101011	86	56	01010110
1	01	00000001	44	2C	00101100	87	57	01010111
2	02	00000010	45	2D	00101101	88	58	01011000
3	03	00000011	46	2E	00101110	89	59	01011001
4	04	00000100	47	2F	00101111	90	5A	01011010
5	05	00000101	48	30	00110000	91	5B	01011011
6	06	00000110	49	31	00110001	92	5C	01011100
7	07	00000111	50	32	00110010	93	5D	01011101
8	08	00001000	51	33	00110011	94	5E	01011110
9	09	00001001	52	34	00110100	95	5F	01011111
10	0A	00001010	53	35	00110101	96	60	01100000
11	0B	00001011	54	36	00110110	97	61	01100001
12	0C	00001100	55	37	00110111	98	62	01100010
13	0D	00001101	56	38	00111000	99	63	01100011
14	0E	00001110	57	39	00111001	100	64	01100100
15	0F	00001111	58	3A	00111010	101	65	01100101
16	10	00010000	59	3B	00111011	102	66	01100110
17	11	00010001	60	3C	00111100	103	67	01100111
18	12	00010010	61	3D	00111101	104	68	01101000
19	13	00010011	62	3E	00111110	105	69	01101001
20	14	00010100	63	3F	00111111	106	6A	01101010
21	15	00010101	64	40	01000000	107	6B	01101011
22	16	00010110	65	41	01000001	108	6C	01101100
23	17	00010111	66	42	01000010	109	6D	01101101
24	18	00011000	67	43	01000011	110	6E	01101110
25	19	00011001	68	44	01000100	111	6F	01101111
26	1A	00011010	69	45	01000101	112	70	01110000
27	1B	00011011	70	46	01000110	113	71	01110001
28	1C	00011100	71	47	01000111	114	72	01110010
29	1D	00011101	72	48	01001000	115	73	01110011
30	1E	00011110	73	49	01001001	116	74	01110100
31	1F	00011111	74	4A	01001010	117	75	01110101
32	20	00100000	75	4B	01001011	118	76	01110110
33	21	00100001	76	4C	01001100	119	77	01110111
34	22	00100010	77	4D	01001101	120	78	01111000
35	23	00100011	78	4E	01001110	121	79	01111001
36	24	00100100	79	4F	01001111	122	7A	01111010
37	25	00100101	80	50	01010000	123	7B	01111011
38	26	00100110	81	51	01010001	124	7C	01111100
39	27	00100111	82	52	01010010	125	7D	01111101
40	28	00101000	83	53	01010011	126	7E	01111110
41	29	00101001	84	54	01010100	127	7F	01111111
42	2A	00101010	85	55	01010101			

FIGURE 6 – Decimal/Hexadecimal/Binary conversion table (integer values from 0 to 127).

DEC	HEX	BIN	DEC	HEX	BIN	DEC	HEX	BIN
128	80	10000000	171	AB	10101011	214	D6	11010110
129	81	10000001	172	AC	10101100	215	D7	11010111
130	82	10000010	173	AD	10101101	216	D8	11011000
131	83	10000011	174	AE	10101110	217	D9	11011001
132	84	10000100	175	AF	10101111	218	DA	11011010
133	85	10000101	176	B0	10110000	219	DB	11011011
134	86	10000110	177	B1	10110001	220	DC	11011100
135	87	10000111	178	B2	10110010	221	DD	11011101
136	88	10001000	179	B3	10110011	222	DE	11011110
137	89	10001001	180	B4	10110100	223	DF	11011111
138	8A	10001010	181	B5	10110101	224	E0	11100000
139	8B	10001011	182	B6	10110110	225	E1	11100001
140	8C	10001100	183	B7	10110111	226	E2	11100010
141	8D	10001101	184	B8	10111000	227	E3	11100011
142	8E	10001110	185	B9	10111001	228	E4	11100100
143	8F	10001111	186	BA	10111010	229	E5	11100101
144	90	10010000	187	BB	10111011	230	E6	11100110
145	91	10010001	188	BC	10111100	231	E7	11100111
146	92	10010010	189	BD	10111101	232	E8	11101000
147	93	10010011	190	BE	10111110	233	E9	11101001
148	94	10010100	191	BF	10111111	234	EA	11101010
149	95	10010101	192	C0	11000000	235	EB	11101011
150	96	10010110	193	C1	11000001	236	EC	11101100
151	97	10010111	194	C2	11000010	237	ED	11101101
152	98	10011000	195	C3	11000011	238	EE	11101110
153	99	10011001	196	C4	11000100	239	EF	11101111
154	9A	10011010	197	C5	11000101	240	F0	11110000
155	9B	10011011	198	C6	11000110	241	F1	11110001
156	9C	10011100	199	C7	11000111	242	F2	11110010
157	9D	10011101	200	C8	11001000	243	F3	11110011
158	9E	10011110	201	C9	11001001	244	F4	11110100
159	9F	10011111	202	CA	11001010	245	F5	11110101
160	A0	10100000	203	CB	11001011	246	F6	11110110
161	A1	10100001	204	CC	11001100	247	F7	11110111
162	A2	10100010	205	CD	11001101	248	F8	11111000
163	A3	10100011	206	CE	11001110	249	F9	11111001
164	A4	10100100	207	CF	11001111	250	FA	11111010
165	A5	10100101	208	D0	11010000	251	FB	11111011
166	A6	10100110	209	D1	11010001	252	FC	11111100
167	A7	10100111	210	D2	11010010	253	FD	11111101
168	A8	10101000	211	D3	11010011	254	FE	11111110
169	A9	10101001	212	D4	11010100	255	FF	11111111
170	AA	10101010	213	D5	11010101			

48685

FIGURE 7 – Decimal/Hexadecimal/Binary conversion table (integer values from 128 to 255).

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

FIGURE 8 – ASCII character conversion table.