

# Advanced Encryption Standard (AES) Block Cipher

# Advanced Encryption Standard - Introduction

The [Advanced Encryption Standard\(AES\)](#) is block cipher that is the replacement for the Data Encryption Standard (DES).

Published by NIST in Nov 2001: FIPS PUB 197 (A copy is available on the UMLearn site).

The actual scheme that the standard is based on is known as [Rijndael](#).

AES has the following key and block lengths:

- block length : 128 bits
- key lengths: 128, 196, and 256 bits

We will consider key lengths of 128 bits. The algorithms for the other key lengths are nearly identical.

# Advanced Encryption Standard - High Level Algorithm

The high-level encryption algorithm is given below.

```
function AESK(M)
  ( $K_0, \dots, K_{10}$ )  $\leftarrow$  expand( $K$ )
   $s \leftarrow M \oplus K_0$ 
  for  $r = 1$  to 10 do
     $s \leftarrow S(s)$ 
     $s \leftarrow \text{shift-rows}(s)$ 
    if  $r \leq 9$  then  $s \leftarrow \text{mix-cols}(s)$  fi
     $s \leftarrow s \oplus K_r$ 
  endfor
  return  $s$ 
```

Figure: AES Encryption Algorithm - High Level

Main Operations:

- expand() - expands the key into 11 keys  $K_0, K_1, \dots, K_{10}$ , each 16 bytes long.
- S() - S-box substitution (also known as SubBytes) of the state
- shift-rows() - shifting rows of the state
- mix-cols() - mixing of the columns (also known as MixColumns)

# The state variable $s$

The variable  $s$  is used to represent the “state” of the encryption process.

Suppose the 16-byte (that is, 128-bit) message  $M = in_0in_1...in_{15}$  where  $in_i$  is a byte (that is, 8 bits) for  $i = 0$  to 15.

The algorithm treats the state as a 4 by 4 array of bytes which is initially set to  $M$  as given in the (left  $4 \times 4$  array of the) following figure.

The algorithms S, shift-rows mix-cols operate on the 4 by 4 state array.

# The state variable $s$

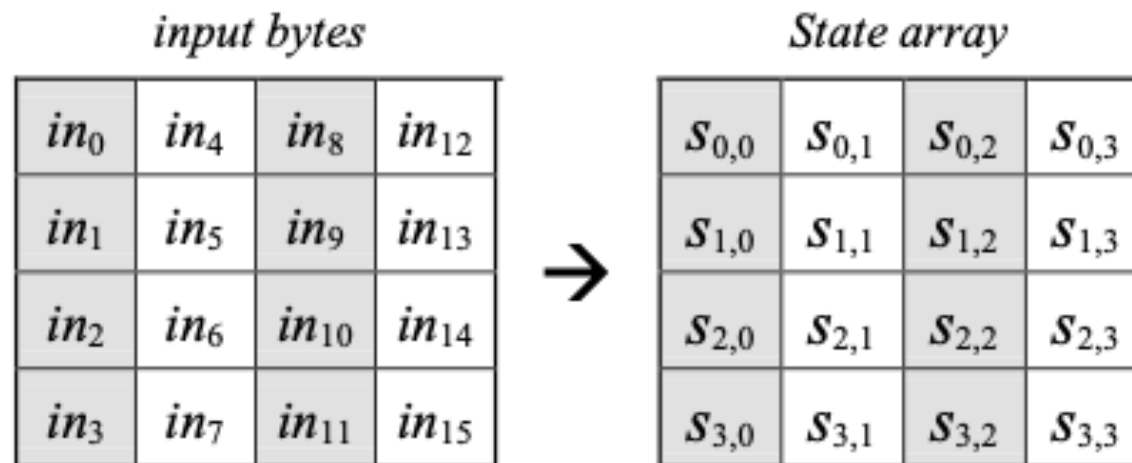


Figure: The mapping of input to state

# The function $S(s)$

The function  $S()$  uses the S-box (provided on following slide) to replace each of the 16 values in the state array  $s$  with a value from the S-box.

We'll represent a byte using two hexadecimal digits enclosed by  $\{$  and  $\}$ .

For example, the byte  $0xfe$  will be written as  $\{fe\}$  (or  $\{FE\}$ ).

Suppose the value in  $s_{i,j} = \{xy\}$ , then  $s_{i,j}$  is replaced with the value in row  $x$  and column  $y$  of the S-box.

For example, if  $s_{1,1} = \{53\}$ , then  $S(s)$  replaces  $s_{1,1}$  with the value  $\{ed\}$ .

Another example: if  $s_{3,1} = \{fd\}$ , then applying  $S(s)$  replaces  $s_{3,1}$  with the value  $\{54\}$ .

# The function $S(s)$

		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

Figure: The AES S-Box

# The function shift — rows( $s$ )

This function cyclicly shifts the rows of  $s$  over a different number of bytes.

The first row is cyclicly shifted to the left by 0 (that is, no shifting).

The second row is cyclicly shifted to the left by 1, third row by 2, and the last row by 3.

The diagram on the next slide illustrates the effect of this function on  $s$ .



# The function shift — rows( $s$ )

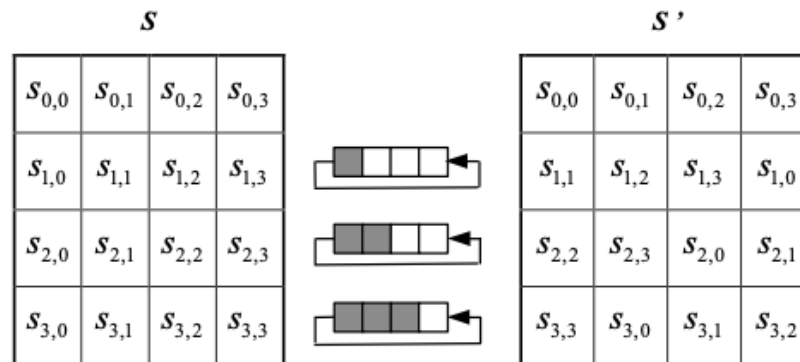


Figure: The shift — rows operation of  $s$

In what follows,  $s$  will denote the current state and  $s'$  will be the next/updated state.

For example,  $s'_{0,2}$  will be updated state value of  $s_{0,2}$  after some operation is performed.

# The function mix – cols( $s$ )

The function mix – cols( $s$ ) operates on the columns of  $s$  one at a time by performing a matrix multiplication in the finite field  $GF(256)$  defined by the irreducible polynomial  $m(x) = x^8 + x^4 + x^3 + x + 1$ .

**Note:** You don't need to understand the details of finite fields to implement AES.

For each column index  $c = 0, 1, 2, 3$  the following matrix operation is applied to column  $c$  of the state array  $s$  to obtain updated values for that column.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

# The function mix – cols( $s$ )

There are two important things to remember when doing the matrix multiplication above:

- 1 The operations (addition and multiplication) are done in the finite field  $GF(2^8)$  with the irreducible polynomial  $m(x) = x^8 + x^4 + x^3 + x + 1$ .
- 2 Each element  $s_{r,c}$  needs to be represented using a polynomial of degree (at most) 7.

Let us now show how addition and multiplication can be done in the finite field (without too much technical mathematics).

# Addition and multiplication in $GF(2^8)$

The elements in the finite field  $GF(2^8)$  are the polynomials of degree 7 with binary coefficients.

**Important:** We can correspond a byte  $b = b_7b_6\dots b_0$  to the polynomial  $b_7x^7 + b_6x^6 + \dots + b_1x^1 + b_0$ , and vice versa.

There is a 1-1 correspondence between elements of  $GF(2^8)$  and the bits values of a 8-bit number.

For example, the byte  $b = \{1b\}$ , having bit values 00011011, corresponds to element  $x^4 + x^3 + x + 1$  in the finite field.

Another example, the element  $x^5 + x^3 + x^2 + 1 \in GF(256)$  corresponds to the the byte  $\{2d\}$ .

We will often switch between the two notations: For example, if we are talking about the field element  $x^5 + x^3 + x + 1$ , we will often denote it by  $\{2b\}$ .

# Addition and multiplication in $GF(2^8)$

Let  $a(x), b(x)$  be two elements in  $GF(2^8)$ . So  $a(x), b(x)$  are polynomials with binary coefficients of degree at most 7.

The addition of  $a(x)$  and  $b(x)$ , denoted by  $a(x) \oplus b(x)$  is obtained by taking the exclusive-or (XOR) of the coefficients of the same degree.

For example, if  $a(x) = x^7 + x^5 + 1$  and  $b(x) = x^6 + x^5 + x^2 + x$ , then  $a(x) \oplus b(x) = x^7 + x^6 + x^2 + x + 1$ .

Note this is exactly the same as taking the binary correspondences of  $a(x), b(x)$ , performing the exclusive-or on the binary values, then corresponding the result back to an element in the field.

For example, if  $a(x) = x^7 + x^5 + 1$  and  $b(x) = x^6 + x^5 + x^2 + x$ , then  $a \oplus b = 10100001 \oplus 01100110 = 11000111$ , which corresponds to  $x^7 + x^6 + x^2 + x^1 + 1$ . That is  $\{a1\} \oplus \{66\} = \{c7\}$ .

# Addition and multiplication in $GF(2^8)$

The process of multiplying two elements in  $GF(2^8)$  is a bit more complicated.

Here, we need to make use of the irreducible polynomial  $m(x) = x^8 + x^4 + x^3 + x + 1$  that was used to define the finite field.

If you take two polynomials  $a(x)$ ,  $b(x)$ , it is possible that when you multiply (denoted by  $a(x) \cdot b(x)$ ) them together, its degree will be greater than 7.

For example, if  $a(x) = x^7 + x^6 + 1$ ,  $b(x) = x$ , then  $a(x)b(x) = x^8 + x^7 + x$ .

In  $GF(256)$ , the  $a(x) \cdot b(x)$  will be its usual product modulo the irreducible polynomial  $m(x) = x^8 + x^4 + x^3 + x + 1$ . That is  $a(x) \cdot b(x) = a(x) \cdot b(x) \pmod{x^8 + x^4 + x^3 + x + 1}$

This can be done using long division.

# Addition and multiplication in $GF(2^8)$

$$\begin{array}{r}
 1 \\
 \hline
 x^8 + x^4 + x^3 + x + 1 \overline{) x^8 + x^7 + x} \\
 \underline{x^8 + x^4 + x^3 + x + 1} \\
 x^7 + x^4 + x^3 + 1
 \end{array}$$

Note:  $x^7 + x^4 + x^3 + 1 \Leftrightarrow 1001\ 1001 = \{99\}$ .

Figure: Long Division


Corresponding back to binary, we see that  $\{c1\} \cdot \{02\} = \{99\}$ .

**Exercise:** Compute  $\{02\} \cdot \{FA\}$  (Answer is  $\{EF\}$ )

# Addition and multiplication in $GF(2^8)$

There is a more efficient way to multiply than to use long division when computing  $\{xy\} \cdot \{02\}$ , where  $x, y$  are hexadecimal digits :

- ① Do a left shift by 1 bit (denoted by  $\lll 1$  here) on  $\{xy\}$  and denote the result by  $c$ .
- ② If the the bit shifted out was a 0, then  $c$  is the final answer.
- ③ If the bit shifted out was a 1, then the final answer is  $c \oplus \{1b\}$ .

For example, to compute  $\{c1\} \cdot \{02\}$  again, we note that  $\{c1\} = 11000001$  so  $\{c1\} \lll 1 = 10000010 = \{82\}$ . Since the bit shifted out was a 1, then  $\{c1\} \cdot \{02\} = \{82\} \oplus \{1b\} = 10000010 \oplus 00011011 = 10011001 = \{99\}$ . 

**Note:** When the MSb is shifted out, a zero bit is shifted into the LSb.



# Addition and multiplication in $GF(2^8)$

Once we know how to multiply by  $\{02\}$ , we can easily multiply by  $\{03\}$  since  $\{03\} = \{02\} \oplus \{01\}$ .

For example, to compute  $\{c1\} \cdot \{03\}$ , this is same as  $\{c1\} \cdot (\{02\} \oplus \{01\}) = (\{c1\} \cdot \{02\}) \oplus (\{c1\} \cdot \{01\}) = \{99\} \oplus \{c1\} = \{58\}$ .

**Exercise:** Compute  $\{03\} \cdot \{FA\}$ .

This is enough to perform the matrix multiplication in the finite field for the mix – cols( $s$ ) function call.

For example,

$$s'_{0,c} = (\{02\} \cdot s_{0,c}) \oplus (\{03\} \cdot s_{1,c}) \oplus (\{01\} \cdot s_{2,c}) \oplus (\{01\} \cdot s_{3,c}).$$

# The $\text{expand}(K)$ Function

The  $\text{expand}(K)$  function takes the input key  $K$  and generates eleven 64-bit keys  $K_0, K_1, \dots, K_{10}$ .

```

function  $\text{expand}(K)$ 
   $K_0 \leftarrow K$ 
  for  $i \leftarrow 1$  to 10 do
     $K_i[0] \leftarrow K_{i-1}[0] \oplus S(K_{i-1}[3] \lll 8) \oplus C_i$ 
     $K_i[1] \leftarrow K_{i-1}[1] \oplus K_i[0]$ 
     $K_i[2] \leftarrow K_{i-1}[2] \oplus K_i[1]$ 
     $K_i[3] \leftarrow K_{i-1}[3] \oplus K_i[2]$ 
  od
  return  $(K_0, \dots, K_{10})$ 

```

Figure: The Key Expansion Algorithm

- Each  $K_i[j]$  is 4 bytes, for  $i = 1$  to 10 and  $j = 0$  to 3.
- $C_1, C_2, \dots, C_{10}$  are the 64-bit constants 0x01000000, 0x02000000, 0x04000000, 0x08000000, 0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x1B000000, 0x36000000, respectively.
- here the  $\lll$  is a cyclic shift (that is, a rotation). This is different from the  $\ll$  used in multiplication in  $GF(256)$ .

# AES Decryption

The decryption algorithm simply runs the encryption algorithm in reverse.

The S, shift-rows, mix-cols functions must be replaced by their inverse operations.

# AES Decryption

As mentioned earlier, you simply run the encryption algorithm backwards.

```
function AESK(M)
  (K0, ..., K10) ← expand(K)
  s ← M ⊕ K0
  for r = 1 to 10 do
    s ← S(s)
    s ← shift-rows(s)
    if r ≤ 9 then s ← mix-cols(s) fi
    s ← s ⊕ Kr
  endfor
  return s
```

Figure: AES Encryption Algorithm -  
High Level

# The Inverse S-Box

The following Inverse S-Box reverses the application of  $S(s)$ .

		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

Figure: The AES Inverse S-Box

# The Inverse (of) shift – rows( $s$ )

The following  $\text{InvShiftRows}(s)$  reverses the application of  $\text{Shift} - \text{Rows}(s)$ .

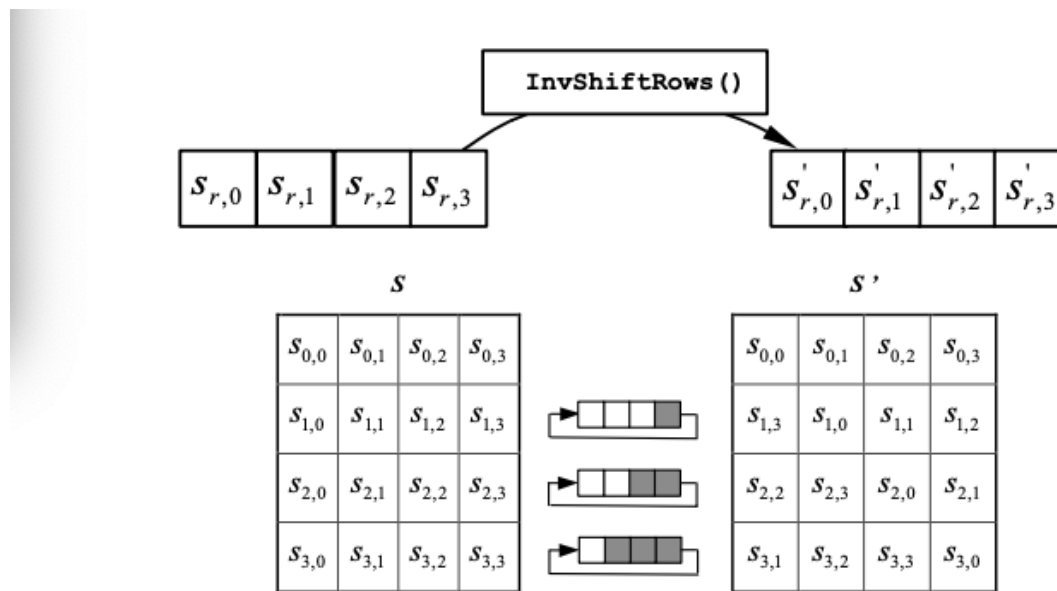



Figure: The Inverse shift – rows operation of  $s$

# The Inverse of mix – cols( $s$ )

To reverse the mix – cols( $s$ ) operation, we need to do another matrix multiplication. 

$$s'_{0,c} = (\{0e\} \cdot s_{0,c}) \oplus (\{0b\} \cdot s_{1,c}) \oplus (\{0d\} \cdot s_{2,c}) \oplus (\{09\} \cdot s_{3,c})$$

$$s'_{1,c} = (\{09\} \cdot s_{0,c}) \oplus (\{0e\} \cdot s_{1,c}) \oplus (\{0b\} \cdot s_{2,c}) \oplus (\{0d\} \cdot s_{3,c})$$

$$s'_{2,c} = (\{0d\} \cdot s_{0,c}) \oplus (\{09\} \cdot s_{1,c}) \oplus (\{0e\} \cdot s_{2,c}) \oplus (\{0b\} \cdot s_{3,c})$$

$$s'_{3,c} = (\{0b\} \cdot s_{0,c}) \oplus (\{0d\} \cdot s_{1,c}) \oplus (\{09\} \cdot s_{2,c}) \oplus (\{0e\} \cdot s_{3,c})$$

**Figure:** The Inverse of mix – cols( $s$ )

How can we do multiplication for by  $\{0e\}$ ,  $\{0b\}$ ,  $\{0d\}$ ,  $\{09\}$ ? It turns out that as long as we know how to multiply by  $\{02\}$ , then that's enough.

# The Inverse of mix – cols( $s$ )

**Example:** Suppose we want to compute  $\{0b\} \cdot \{1e\}$ .

Since  $\{0b\} = \{08\} \oplus \{02\} \oplus \{01\}$ , then



$$\{0b\} \cdot \{1e\} = (\{01\} \cdot \{1e\}) \oplus (\{02\} \cdot \{1e\}) \oplus (\{08\} \cdot \{1e\}).$$

Now,

$$\{01\} \cdot \{1e\} = \{1e\}$$

$$\{02\} \cdot \{1e\} = \{3c\} \text{ (check!)}$$

$$\{04\} \cdot \{1e\} = \{02\} \cdot (\{02\} \cdot \{1e\}) = \{02\} \cdot \{3c\} = \{78\} \text{ (check!)}$$

$$\{08\} \cdot \{1e\} = \{02\} \cdot (\{04\} \cdot \{1e\}) = \{02\} \cdot \{78\} = \{F0\}.$$

Therefore  $\{0b\} \cdot \{1e\} = \{1e\} \oplus \{3c\} \oplus \{F0\} = \{D2\}$ .



# The Inverse of mix – cols(*s*)

**Example:** Suppose we want to compute  $\{0e\} \cdot \{34\}$ .

Since  $\{0e\} = \{08\} \oplus \{04\} \oplus \{02\}$ , then



$$\{0e\} \cdot \{34\} = \{02\} \cdot \{34\} \oplus \{04\} \cdot \{34\} \oplus \{08\} \cdot \{34\}.$$

$$\{01\} \cdot \{34\} = \{34\}$$

$$\{02\} \cdot \{34\} = \{68\} \text{ (check!)}$$

$$\{04\} \cdot \{34\} = \{02\} \cdot (\{02\} \cdot \{34\}) = \{02\} \cdot \{68\} = \{d0\} \text{ (check!)}$$

$$\{08\} \cdot \{34\} = \{02\} \cdot (\{04\} \cdot \{34\}) = \{02\} \cdot \{d0\} = \{bb\}.$$



When we do a shift (left 1 bit) to compute  $\{02\} \cdot \{d0\}$ , we get  $\{a0\}$  and there was a carry-out bit with value 1. Therefore, we need to perform  $\{a0\} \oplus \{1b\}$  to get the correct answer  $\{bb\}$ .

Therefore  $\{0e\} \cdot \{34\} = \{68\} \oplus \{d0\} \oplus \{bb\}$ .

# The Inverse of mix – $\text{cols}(s)$

Therefore, by doing various (field) multiplications by  $\{02\}$  and then (field) adding up the required results, we can multiply any two elements in  $GF(2^8)$ .

This can be easily implemented.

Don't forget that in the process of multiplying by 2, if you shifted out a 1 bit, you need to XOR the shifted result with the value  $\{1b\}$ .

The FIPS 197 document provides several examples that you should look at.

# AES Security

Subjected to intense analysis during the AES selection process and afterwards.

No known practical cryptanalytic attacks have been found that are significantly better than an exhaustive search for the key.

Best known attack, known as the [biclique attack](#) (2011) reduces the complexity of an exhaustive search by a factor of 4 or 5.

Secured against differential and linear attacks that have been used on DES.