# How to implement 128 bit AES / Rijndael from scratch: A simple Ruby implementation with lots of comments

By Chris Hulbert - [chris.hulbert@gmail.com](mailto:chris.hulbert@gmail.com) - <http://github.com/chrishulbert/crypto>

As a learning exercise recently, I set about implementing a bunch of crypto algorithms from scratch, in such a way as to learn how they work, rather than simply cut and pasting some open source code. In trying to learn and implement AES, I found there’s no central document that details how to do this, and had to cobble it together. This document is my attempt to improve that situation.

Keep in mind that this implementation is more aimed at learning how the algorithm works, rather than implementing it in an optimised way, because I often find that these optimisations make it harder to understand what’s really going on. So, if you keep in mind that this code isn’t production-ready, we should get along just fine.

# AES quick summary

AES is a form of Rijndael, invented by Joan Daemen and Vincent Rijmen, limited to certain block and key sizes. So for most purposes, they’re the same thing. Here, we’ll implement the 128 bit form. The general gist is to take a key and a message, expand the key to create 11 round keys, and perform 11 rounds of processing on the message to produce either the cipher text (when encrypting) or the original text (when decrypting).

Now for some handy functions we’ll use a lot, but aren’t really related to AES in particular:

class String

def to\_bytes # Converts a hex string into an array of bytes

(0...self.length/2).map {|i|self[i\*2,2].to\_i(16)}

end

end

class Array

def pretty # Converts an array into a nicely formatted string

self.map{|v|"%02X"%v}.join('-')

end

end

# Expanding the key

Firstly, read this to get an overview of what is involved in expanding the key: <http://en.wikipedia.org/wiki/Rijndael_key_schedule>

Here’s a function that takes a 128-bit key in the form of an array of 16 bytes (integers) and returns an array of 176 bytes. The comments are mostly copied from the pseudocode on the above Wiki page under the ‘Key schedule description’ section, except for skipping the parts for > 128 bit keys:

class Array

def expand\_key # http://en.wikipedia.org/wiki/Rijndael\_key\_schedule#The\_key\_schedule

n=16 # n has a value of 16 for 128-bit keys, 24 for 192-bit keys, and 32 for 256-bit keys

b=176 # b has a value of 176 for 128-bit keys, 208 for 192-bit keys, and 240 for 256-bit keys

key=self[0,n] # The first n bytes of the expanded key are simply the encryption key

i=1 # The rcon iteration value i is set to 1

until key.length==b do # Until we have b bytes of expanded key, we do the following:

t=key[-4,4] # We assign the value of the previous four bytes in the expanded key to t

t=t.key\_schedule\_core(i) # We perform the key schedule core on t, with i as the rcon iteration value

i+=1 # We increment i by 1

t=t.xor(key[-n,4]) # We exclusive-or t with the four-byte block n bytes before the new expanded key.

key+=t # This becomes the next 4 bytes in the expanded key

3.times { # We then do the following three times to create the next twelve bytes

t=key[-4,4] # We assign the value of the previous 4 bytes in the expanded key to t

t=t.xor(key[-n,4]) # We exclusive-or t with the four-byte block n bytes before

key+=t # This becomes the next 4 bytes in the expanded key

}

end

key

end

end

Now to use this function, we first have to implement a few other functions it uses first. First up, here’s the xor function, this one simply xor’s the current array byte by byte with another one:

class Array

def xor(other)

out=[]

zip(other){|a,b| out << (a^b)}

out

end

end

And, more importantly, here’s the key schedule core function, with comments following the description in the Wiki:

class Array

def key\_schedule\_core(i) # The input is a 4-byte array and an iteration number i. The output is a 32-bit word.

o=self[1,3]+self[0,1] # Rotate the output eight bits to the left

o=o.sub\_bytes # Apply Rijndael's S-box on all four individual bytes in the output word

o[0]^=i.rcon # On just the first (leftmost) byte of the output word, perform the rcon operation with i

# as the input, and exclusive or the rcon output with the first byte of the output word

o

end

end

Now the sub\_bytes function’s job is to apply the substitution box lookup table to each of the bytes. These lookup tables are generated through complicated mathematical algorithms, but the simplest thing is to hard code them. I’m creating a function against the Integer class so that any byte can return it’s result from the s-box table, and the sub\_bytes function so that arrays can perform this lookup for each of their items:

class Array

def sub\_bytes # Substitute all bytes in this array with their s-box result

map{|i|i.sbox}

end

end

class Integer

def sbox # http://en.wikipedia.org/wiki/Rijndael\_S-box

[0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,

0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,

0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,

0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,

0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,

0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,

0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,

0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,

0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,

0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,

0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,

0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,

0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,

0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,

0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,

0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16][self]

end

end

Also, while we’re on the topic, we’ll create the inverse s-box and sub\_bytes functions, which are needed for decryption later:

class Array

def sub\_bytes\_inv

map{|i|i.sbox\_inv}

end

end

class Integer

def sbox\_inv # http://en.wikipedia.org/wiki/Rijndael\_S-box

[0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb,

0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb,

0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e,

0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25,

0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92,

0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84,

0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06,

0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b,

0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xe6,0x73,

0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e,

0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b,

0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0,0xfe,0x78,0xcd,0x5a,0xf4,

0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10,0x59,0x27,0x80,0xec,0x5f,

0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,0x2d,0xe5,0x7a,0x9f,0x93,0xc9,0x9c,0xef,

0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61,

0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14,0x63,0x55,0x21,0x0c,0x7d][self]

end

end

We also need the Rcon function. This, again (like a lot of functions), involves a precalculated lookup table. The table in Wiki is a lot longer, but we only need the first line of values:

class Integer

def rcon # http://en.wikipedia.org/wiki/Rijndael\_key\_schedule#Rcon

[0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a][self]

end

end

Now we’ve implemented the key expander, let’s test it:

k="12345612345612345612345612345612".to\_bytes

puts k.expand\_key.pretty

If you run this, you should get the following results:

12-34-56-12-34-56-12-34-56-12-34-56-12-34-56-12-

0B-85-9F-DB-3F-D3-8D-EF-69-C1-B9-B9-7B-F5-EF-AB-

EF-5A-FD-FA-D0-89-70-15-B9-48-C9-AC-C2-BD-26-07-

91-AD-38-DF-41-24-48-CA-F8-6C-81-66-3A-D1-A7-61-

A7-F1-D7-5F-E6-D5-9F-95-1E-B9-1E-F3-24-68-B9-92-

F2-A7-98-69-14-72-07-FC-0A-CB-19-0F-2E-A3-A0-9D-

D8-47-C6-58-CC-35-C1-A4-C6-FE-D8-AB-E8-5D-78-36-

D4-FB-C3-C3-18-CE-02-67-DE-30-DA-CC-36-6D-A2-FA-

68-C1-EE-C6-70-0F-EC-A1-AE-3F-36-6D-98-52-94-97-

73-E3-66-80-03-EC-8A-21-AD-D3-BC-4C-35-81-28-DB-

49-D7-DF-16-4A-3B-55-37-E7-E8-E9-7B-D2-69-C1-A0

This data is your 11 round keys, and you’re ready for the encryption (and decryption) process.

# Encoding

AES encryption is described here: <http://en.wikipedia.org/wiki/Advanced_Encryption_Standard>. Basically, it works in a series of rounds, which looks like this:

* Key expansion
* First round:
  + Xor round key
* 9 middle rounds:
  + Substitute bytes
  + Shift rows
  + Mix columns
  + Xor round Key
* Last round:
  + Substitute bytes
  + Shift rows
  + Xor round Key

And so, our encryption function looks like this:

def encrypt(m,k)

# Key expansion

k = k.expand\_key

# First Round

c = m.xor\_round\_key(k,0)

# Middle rounds

9.times {|i|

c = c.sub\_bytes

c = c.shift\_rows

c = c.mix\_cols

c = c.xor\_round\_key(k,i+1)

}

# Final Round

c = c.sub\_bytes

c = c.shift\_rows

c = c.xor\_round\_key(k,10)

end

Looks pretty simple so far, right? Well, we now have to implement the shift rows, mix columns, and xor round keys functions.

## Xor Round Key

This function is pretty straight forwards. If you recall, in the key expansion function we generate a 176 byte expanded key. This happens to be exactly 11 round keys, given that each key is 16 bytes (128 bits). So this function takes the 176-byte expanded key, and the number of the round key from 0 to 10 inclusive, and xor’s the correct round key with the current state (stored in the ‘c’ variable above):

class Array

def xor\_round\_key(keys,i)

xor(keys[i\*16,16])

end

end

## Shift Rows

The shift rows step can be seen here: <http://en.wikipedia.org/wiki/Advanced_Encryption_Standard#The_ShiftRows_step>

Think of the 16 bytes of the state as a 4x4 matrix running downwards then across. This is illustrated below, where the numbers represent the offset within the 16-byte array:

0 4 8 12

1 5 9 13

2 6 10 14

3 7 11 15

To perform the shift rows, the top row of this matrix stays put, the second row is shifted left one byte, the third row is shifted left two bytes, and the bottom row is shifted left three bytes. We want the following result matrix, where the numbers represent the offsets from the original matrix:

0 4 8 12

5 9 13 1 <- Shifted left 1 place

10 14 2 6 <- Shifted left 2 places

15 3 7 11 <- Shifted left 3 places

While we’re at it, let’s do the inverse matrix, which we’ll need for decryption later on:

0 4 8 12

13 1 5 9 <- Shifted right 1 place

10 14 2 6 <- Shifted right 2 places

7 11 15 3 <- Shifted right 3 places

Now, remembering that this matrix represents the 16-byte array in a downwards then across arrangement, we convert the two results matrices above into the below array representation:

0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1, 6, 11 <- Shift left (normal, for encryption)

0, 13, 10, 7, 4, 1, 14, 11, 8, 5, 2, 15, 12, 9, 6, 3 <- Shift right (inverse, for decryption)

So we can make ruby functions to rearrange the 16-byte state according to the above offsets like this:

class Array

def shift\_rows

[0,5,10,15,4,9,14,3,8,13,2,7,12,1,6,11].map{|i|self[i]}

end

def shift\_rows\_inv

[0,13,10,7,4,1,14,11,8,5,2,15,12,9,6,3].map{|i|self[i]}

end

end

## Mix Columns

You can read about this step here: <http://en.wikipedia.org/wiki/Rijndael_mix_columns>. There’s a couple of complicated looking matrices there, but it’s not too bad in implementation. Firstly, we need to split out 16-byte state into 4 columns of 4 bytes. Each of these columns has the mix column function called, and their results are joined to create the final result. In code, this process looks like this for both the normal and inverse functions:

class Array

def mix\_cols

self[0,4].mix\_col +

self[4,4].mix\_col +

self[8,4].mix\_col +

self[12,4].mix\_col

end

def mix\_cols\_inv

self[0,4].mix\_col\_inv +

self[4,4].mix\_col\_inv +

self[8,4].mix\_col\_inv +

self[12,4].mix\_col\_inv

end

end

Now we need to create the functions to perform the matrix calculation agains each column of 4 bytes. In the Wiki article, you can see the following formula:

r0 = 2a0 + 3a1 + 1a2 + 1a3

r1 = 1a0 + 2a1 + 3a2 + 1a3

r2 = 1a0 + 1a1 + 2a2 + 3a3

r3 = 3a0 + 1a1 + 1a2 + 2a3

In this formula, a0..a3 is equivalent to the 4 input bytes that make up the column, and r0..r3 are the resultant output bytes. Plus (+) is actually implemented as an Xor, due to the formula being performed within the Galois field: <http://en.wikipedia.org/wiki/Rijndael_Galois_field>. And multiplication by 2 and 3 are also complicated Galois operations, but you can simply make lookup tables for these. And so the implementation for the normal (encryption) mix column is:

class Array

def mix\_col # http://en.wikipedia.org/wiki/Rijndael\_mix\_columns

a0,a1,a2,a3 = self

[ a0.g2 ^ a1.g3 ^ a2 ^ a3, # r0 = 2a0 + 3a1 + 1a2 + 1a3

a0 ^ a1.g2 ^ a2.g3 ^ a3, # r1 = 1a0 + 2a1 + 3a2 + 1a3

a0 ^ a1 ^ a2.g2 ^ a3.g3, # r2 = 1a0 + 1a1 + 2a2 + 3a3

a0.g3 ^ a1 ^ a2 ^ a3.g2] # r3 = 3a0 + 1a1 + 1a2 + 2a3

end

end

The lookup tables for the galois multiplication by 2 and 3 are below:

class Integer

def g2 # Perform a galois field multiplication of the current value by 2

# Lookup table from here: http://code.google.com/p/byte-oriented-aes/

[0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,

0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,

0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,

0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,

0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,

0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,

0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,

0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,

0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,

0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,

0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,

0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,

0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,

0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,

0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,

0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5][self]

end

def g3

[0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,

0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,

0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,

0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,

0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,

0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,

0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,

0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,

0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,

0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,

0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xe9,0xea,

0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,

0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,

0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,

0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,

0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a][self]

end

end

Now let’s test this. Under the ‘Test vectors’ section on the Wiki page, let’s run mix column function on the ‘before’ column, and see if we get the ‘after’ values:

puts "db135345".to\_bytes.mix\_col.pretty # Should output: 8e4da1bc

puts "f20a225c".to\_bytes.mix\_col.pretty # Should output: 9fdc589d

puts "01010101".to\_bytes.mix\_col.pretty # Should output: 01010101

puts "c6c6c6c6".to\_bytes.mix\_col.pretty # Should output: c6c6c6c6

puts "d4d4d4d5".to\_bytes.mix\_col.pretty # Should output: d5d5d7d6

puts "2d26314c".to\_bytes.mix\_col.pretty # Should output: 4d7ebdf8

If that’s working for you, let’s move on to implementing the inverse mix columns, which we’ll need for decrypting. Now the formula on the wiki page for applying the inverse matrix to a 4-byte row looks like this:

r0 = 14a0 + 9a3 + 13a2 + 11a1

r1 = 14a1 + 9a0 + 13a3 + 11a2

r2 = 14a2 + 9a1 + 13a0 + 11a3

r3 = 14a3 + 9a2 + 13a1 + 11a0

This becomes the following code:

class Array

def mix\_col\_inv # http://en.wikipedia.org/wiki/Rijndael\_mix\_columns

a0,a1,a2,a3 = self

[ a0.g14 ^ a3.g9 ^ a2.g13 ^ a1.g11, # r0 = 14a0 + 9a3 + 13a2 + 11a1

a1.g14 ^ a0.g9 ^ a3.g13 ^ a2.g11, # r1 = 14a1 + 9a0 + 13a3 + 11a2

a2.g14 ^ a1.g9 ^ a0.g13 ^ a3.g11, # r2 = 14a2 + 9a1 + 13a0 + 11a3

a3.g14 ^ a2.g9 ^ a1.g13 ^ a0.g11] # r3 = 14a3 + 9a2 + 13a1 + 11a0

end

end

Notice that we now have to be able to multiply be 14, 9, 13, and 11. To do this, we’ll need 4 more lookup tables:

class Integer

def g9 # Galois multiply by 9

[0x00,0x09,0x12,0x1b,0x24,0x2d,0x36,0x3f,0x48,0x41,0x5a,0x53,0x6c,0x65,0x7e,0x77,

0x90,0x99,0x82,0x8b,0xb4,0xbd,0xa6,0xaf,0xd8,0xd1,0xca,0xc3,0xfc,0xf5,0xee,0xe7,

0x3b,0x32,0x29,0x20,0x1f,0x16,0x0d,0x04,0x73,0x7a,0x61,0x68,0x57,0x5e,0x45,0x4c,

0xab,0xa2,0xb9,0xb0,0x8f,0x86,0x9d,0x94,0xe3,0xea,0xf1,0xf8,0xc7,0xce,0xd5,0xdc,

0x76,0x7f,0x64,0x6d,0x52,0x5b,0x40,0x49,0x3e,0x37,0x2c,0x25,0x1a,0x13,0x08,0x01,

0xe6,0xef,0xf4,0xfd,0xc2,0xcb,0xd0,0xd9,0xae,0xa7,0xbc,0xb5,0x8a,0x83,0x98,0x91,

0x4d,0x44,0x5f,0x56,0x69,0x60,0x7b,0x72,0x05,0x0c,0x17,0x1e,0x21,0x28,0x33,0x3a,

0xdd,0xd4,0xcf,0xc6,0xf9,0xf0,0xeb,0xe2,0x95,0x9c,0x87,0x8e,0xb1,0xb8,0xa3,0xaa,

0xec,0xe5,0xfe,0xf7,0xc8,0xc1,0xda,0xd3,0xa4,0xad,0xb6,0xbf,0x80,0x89,0x92,0x9b,

0x7c,0x75,0x6e,0x67,0x58,0x51,0x4a,0x43,0x34,0x3d,0x26,0x2f,0x10,0x19,0x02,0x0b,

0xd7,0xde,0xc5,0xcc,0xf3,0xfa,0xe1,0xe8,0x9f,0x96,0x8d,0x84,0xbb,0xb2,0xa9,0xa0,

0x47,0x4e,0x55,0x5c,0x63,0x6a,0x71,0x78,0x0f,0x06,0x1d,0x14,0x2b,0x22,0x39,0x30,

0x9a,0x93,0x88,0x81,0xbe,0xb7,0xac,0xa5,0xd2,0xdb,0xc0,0xc9,0xf6,0xff,0xe4,0xed,

0x0a,0x03,0x18,0x11,0x2e,0x27,0x3c,0x35,0x42,0x4b,0x50,0x59,0x66,0x6f,0x74,0x7d,

0xa1,0xa8,0xb3,0xba,0x85,0x8c,0x97,0x9e,0xe9,0xe0,0xfb,0xf2,0xcd,0xc4,0xdf,0xd6,

0x31,0x38,0x23,0x2a,0x15,0x1c,0x07,0x0e,0x79,0x70,0x6b,0x62,0x5d,0x54,0x4f,0x46][self]

end

def g11

[0x00,0x0b,0x16,0x1d,0x2c,0x27,0x3a,0x31,0x58,0x53,0x4e,0x45,0x74,0x7f,0x62,0x69,

0xb0,0xbb,0xa6,0xad,0x9c,0x97,0x8a,0x81,0xe8,0xe3,0xfe,0xf5,0xc4,0xcf,0xd2,0xd9,

0x7b,0x70,0x6d,0x66,0x57,0x5c,0x41,0x4a,0x23,0x28,0x35,0x3e,0x0f,0x04,0x19,0x12,

0xcb,0xc0,0xdd,0xd6,0xe7,0xec,0xf1,0xfa,0x93,0x98,0x85,0x8e,0xbf,0xb4,0xa9,0xa2,

0xf6,0xfd,0xe0,0xeb,0xda,0xd1,0xcc,0xc7,0xae,0xa5,0xb8,0xb3,0x82,0x89,0x94,0x9f,

0x46,0x4d,0x50,0x5b,0x6a,0x61,0x7c,0x77,0x1e,0x15,0x08,0x03,0x32,0x39,0x24,0x2f,

0x8d,0x86,0x9b,0x90,0xa1,0xaa,0xb7,0xbc,0xd5,0xde,0xc3,0xc8,0xf9,0xf2,0xef,0xe4,

0x3d,0x36,0x2b,0x20,0x11,0x1a,0x07,0x0c,0x65,0x6e,0x73,0x78,0x49,0x42,0x5f,0x54,

0xf7,0xfc,0xe1,0xea,0xdb,0xd0,0xcd,0xc6,0xaf,0xa4,0xb9,0xb2,0x83,0x88,0x95,0x9e,

0x47,0x4c,0x51,0x5a,0x6b,0x60,0x7d,0x76,0x1f,0x14,0x09,0x02,0x33,0x38,0x25,0x2e,

0x8c,0x87,0x9a,0x91,0xa0,0xab,0xb6,0xbd,0xd4,0xdf,0xc2,0xc9,0xf8,0xf3,0xee,0xe5,

0x3c,0x37,0x2a,0x21,0x10,0x1b,0x06,0x0d,0x64,0x6f,0x72,0x79,0x48,0x43,0x5e,0x55,

0x01,0x0a,0x17,0x1c,0x2d,0x26,0x3b,0x30,0x59,0x52,0x4f,0x44,0x75,0x7e,0x63,0x68,

0xb1,0xba,0xa7,0xac,0x9d,0x96,0x8b,0x80,0xe9,0xe2,0xff,0xf4,0xc5,0xce,0xd3,0xd8,

0x7a,0x71,0x6c,0x67,0x56,0x5d,0x40,0x4b,0x22,0x29,0x34,0x3f,0x0e,0x05,0x18,0x13,

0xca,0xc1,0xdc,0xd7,0xe6,0xed,0xf0,0xfb,0x92,0x99,0x84,0x8f,0xbe,0xb5,0xa8,0xa3][self]

end

def g13

[0x00,0x0d,0x1a,0x17,0x34,0x39,0x2e,0x23,0x68,0x65,0x72,0x7f,0x5c,0x51,0x46,0x4b,

0xd0,0xdd,0xca,0xc7,0xe4,0xe9,0xfe,0xf3,0xb8,0xb5,0xa2,0xaf,0x8c,0x81,0x96,0x9b,

0xbb,0xb6,0xa1,0xac,0x8f,0x82,0x95,0x98,0xd3,0xde,0xc9,0xc4,0xe7,0xea,0xfd,0xf0,

0x6b,0x66,0x71,0x7c,0x5f,0x52,0x45,0x48,0x03,0x0e,0x19,0x14,0x37,0x3a,0x2d,0x20,

0x6d,0x60,0x77,0x7a,0x59,0x54,0x43,0x4e,0x05,0x08,0x1f,0x12,0x31,0x3c,0x2b,0x26,

0xbd,0xb0,0xa7,0xaa,0x89,0x84,0x93,0x9e,0xd5,0xd8,0xcf,0xc2,0xe1,0xec,0xfb,0xf6,

0xd6,0xdb,0xcc,0xc1,0xe2,0xef,0xf8,0xf5,0xbe,0xb3,0xa4,0xa9,0x8a,0x87,0x90,0x9d,

0x06,0x0b,0x1c,0x11,0x32,0x3f,0x28,0x25,0x6e,0x63,0x74,0x79,0x5a,0x57,0x40,0x4d,

0xda,0xd7,0xc0,0xcd,0xee,0xe3,0xf4,0xf9,0xb2,0xbf,0xa8,0xa5,0x86,0x8b,0x9c,0x91,

0x0a,0x07,0x10,0x1d,0x3e,0x33,0x24,0x29,0x62,0x6f,0x78,0x75,0x56,0x5b,0x4c,0x41,

0x61,0x6c,0x7b,0x76,0x55,0x58,0x4f,0x42,0x09,0x04,0x13,0x1e,0x3d,0x30,0x27,0x2a,

0xb1,0xbc,0xab,0xa6,0x85,0x88,0x9f,0x92,0xd9,0xd4,0xc3,0xce,0xed,0xe0,0xf7,0xfa,

0xb7,0xba,0xad,0xa0,0x83,0x8e,0x99,0x94,0xdf,0xd2,0xc5,0xc8,0xeb,0xe6,0xf1,0xfc,

0x67,0x6a,0x7d,0x70,0x53,0x5e,0x49,0x44,0x0f,0x02,0x15,0x18,0x3b,0x36,0x21,0x2c,

0x0c,0x01,0x16,0x1b,0x38,0x35,0x22,0x2f,0x64,0x69,0x7e,0x73,0x50,0x5d,0x4a,0x47,

0xdc,0xd1,0xc6,0xcb,0xe8,0xe5,0xf2,0xff,0xb4,0xb9,0xae,0xa3,0x80,0x8d,0x9a,0x97][self]

end

def g14

[0x00,0x0e,0x1c,0x12,0x38,0x36,0x24,0x2a,0x70,0x7e,0x6c,0x62,0x48,0x46,0x54,0x5a,

0xe0,0xee,0xfc,0xf2,0xd8,0xd6,0xc4,0xca,0x90,0x9e,0x8c,0x82,0xa8,0xa6,0xb4,0xba,

0xdb,0xd5,0xc7,0xc9,0xe3,0xed,0xff,0xf1,0xab,0xa5,0xb7,0xb9,0x93,0x9d,0x8f,0x81,

0x3b,0x35,0x27,0x29,0x03,0x0d,0x1f,0x11,0x4b,0x45,0x57,0x59,0x73,0x7d,0x6f,0x61,

0xad,0xa3,0xb1,0xbf,0x95,0x9b,0x89,0x87,0xdd,0xd3,0xc1,0xcf,0xe5,0xeb,0xf9,0xf7,

0x4d,0x43,0x51,0x5f,0x75,0x7b,0x69,0x67,0x3d,0x33,0x21,0x2f,0x05,0x0b,0x19,0x17,

0x76,0x78,0x6a,0x64,0x4e,0x40,0x52,0x5c,0x06,0x08,0x1a,0x14,0x3e,0x30,0x22,0x2c,

0x96,0x98,0x8a,0x84,0xae,0xa0,0xb2,0xbc,0xe6,0xe8,0xfa,0xf4,0xde,0xd0,0xc2,0xcc,

0x41,0x4f,0x5d,0x53,0x79,0x77,0x65,0x6b,0x31,0x3f,0x2d,0x23,0x09,0x07,0x15,0x1b,

0xa1,0xaf,0xbd,0xb3,0x99,0x97,0x85,0x8b,0xd1,0xdf,0xcd,0xc3,0xe9,0xe7,0xf5,0xfb,

0x9a,0x94,0x86,0x88,0xa2,0xac,0xbe,0xb0,0xea,0xe4,0xf6,0xf8,0xd2,0xdc,0xce,0xc0,

0x7a,0x74,0x66,0x68,0x42,0x4c,0x5e,0x50,0x0a,0x04,0x16,0x18,0x32,0x3c,0x2e,0x20,

0xec,0xe2,0xf0,0xfe,0xd4,0xda,0xc8,0xc6,0x9c,0x92,0x80,0x8e,0xa4,0xaa,0xb8,0xb6,

0x0c,0x02,0x10,0x1e,0x34,0x3a,0x28,0x26,0x7c,0x72,0x60,0x6e,0x44,0x4a,0x58,0x56,

0x37,0x39,0x2b,0x25,0x0f,0x01,0x13,0x1d,0x47,0x49,0x5b,0x55,0x7f,0x71,0x63,0x6d,

0xd7,0xd9,0xcb,0xc5,0xef,0xe1,0xf3,0xfd,0xa7,0xa9,0xbb,0xb5,0x9f,0x91,0x83,0x8d][self]

end

end

Let’s test the inverse mix column function now, again using the values from the test vectors on the Wiki but starting with the ‘After’ values and trying the produce the ‘Before’ values:

puts "8e4da1bc".to\_bytes.mix\_col\_inv.pretty # Should output: db135345

puts "9fdc589d".to\_bytes.mix\_col\_inv.pretty # Should output: f20a225c

puts "01010101".to\_bytes.mix\_col\_inv.pretty # Should output: 01010101

puts "c6c6c6c6".to\_bytes.mix\_col\_inv.pretty # Should output: c6c6c6c6

puts "d5d5d7d6".to\_bytes.mix\_col\_inv.pretty # Should output: d4d4d4d5

puts "4d7ebdf8".to\_bytes.mix\_col\_inv.pretty # Should output: 2d26314c

## Testing Encryption

Ok now you should have enough code for the encryption to work. Let’s test it:

m="abcdefabcdefabcdefabcdefabcdefab".to\_bytes

k="12345612345612345612345612345612".to\_bytes

c=encrypt(m,k)

puts c.pretty # Should be 85E5A3D7356A61E29A8AFA559AD67102

You should get the following result:

85-E5-A3-D7-35-6A-61-E2-9A-8A-FA-55-9A-D6-71-02

I’ve found a neat online calculator for testing these calculations: <http://www.unsw.adfa.edu.au/~lpb/src/AEScalc/AEScalc.html>

Decryption  
Decryption is simply the reverse order of encryption, with the inverse functions used where appropriate:

def decrypt(c,k)

# Key expansion

k = k.expand\_key

# Reverse the final round

c = c.xor\_round\_key(k,10)

c = c.shift\_rows\_inv

c = c.sub\_bytes\_inv

# Reverse the middle rounds

9.times {|i|

c = c.xor\_round\_key(k,9-i)

c = c.mix\_cols\_inv

c = c.shift\_rows\_inv

c = c.sub\_bytes\_inv

}

# Reverse the first round

m = c.xor\_round\_key(k,0)

end

Notice that the only function that doesn’t need an inverse is the xor\_round\_key, by virtue of the nature of Xoring, two Xor’s will undo each other. Ok now let’s test the decrypt out:

c="85E5A3D7356A61E29A8AFA559AD67102".to\_bytes

k="12345612345612345612345612345612".to\_bytes

m=decrypt(c,k)

puts m.pretty # Should be abcdefabcdefabcdefabcdefabcdefab

You should get the following result:

AB-CD-EF-AB-CD-EF-AB-CD-EF-AB-CD-EF-AB-CD-EF-AB

That’s it! Thanks for reading. I hope you got an understanding of the internals of how AES works. Feel free to send me an email if this has helped you in any way.

Chris Hulbert

[chris.hulbert@gmail.com](mailto:chris.hulbert@gmail.com)

# Complete Ruby source code

class String

def to\_bytes # Converts a hex string into an array of bytes

(0...self.length/2).map {|i|self[i\*2,2].to\_i(16)}

end

end

class Array

def pretty # Converts an array into a nicely formatted string

self.map{|v|"%02X"%v}.join('-')

end

end

class Array

def expand\_key # http://en.wikipedia.org/wiki/Rijndael\_key\_schedule#The\_key\_schedule

n=16 # n has a value of 16 for 128-bit keys, 24 for 192-bit keys, and 32 for 256-bit keys

b=176 # b has a value of 176 for 128-bit keys, 208 for 192-bit keys, and 240 for 256-bit keys

key=self[0,n] # The first n bytes of the expanded key are simply the encryption key

i=1 # The rcon iteration value i is set to 1

until key.length==b do # Until we have b bytes of expanded key, we do the following:

t=key[-4,4] # We assign the value of the previous four bytes in the expanded key to t

t=t.key\_schedule\_core(i) # We perform the key schedule core on t, with i as the rcon iteration value

i+=1 # We increment i by 1

t=t.xor(key[-n,4]) # We exclusive-or t with the four-byte block n bytes before the new expanded key.

key+=t # This becomes the next 4 bytes in the expanded key

3.times { # We then do the following three times to create the next twelve bytes

t=key[-4,4] # We assign the value of the previous 4 bytes in the expanded key to t

t=t.xor(key[-n,4]) # We exclusive-or t with the four-byte block n bytes before

key+=t # This becomes the next 4 bytes in the expanded key

}

end

key

end

end

class Array

def xor(other)

out=[]

zip(other){|a,b| out << (a^b)}

out

end

end

class Array

def key\_schedule\_core(i) # The input is a 4-byte array and an iteration number i. The output is a 32-bit word.

o=self[1,3]+self[0,1] # Rotate the output eight bits to the left

o=o.sub\_bytes # Apply Rijndael's S-box on all four individual bytes in the output word

o[0]^=i.rcon # On just the first (leftmost) byte of the output word, perform the rcon operation with i

# as the input, and exclusive or the rcon output with the first byte of the output word

o

end

end

class Array

def sub\_bytes # Substitute all bytes in this array with their s-box result

map{|i|i.sbox}

end

end

class Integer

def sbox # http://en.wikipedia.org/wiki/Rijndael\_S-box

[0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,

0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,

0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,

0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,

0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,

0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,

0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,

0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,

0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,

0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,

0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,

0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,

0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,

0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,

0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,

0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16][self]

end

end

class Array

def sub\_bytes\_inv

map{|i|i.sbox\_inv}

end

end

class Integer

def sbox\_inv # http://en.wikipedia.org/wiki/Rijndael\_S-box

[0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb,

0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb,

0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e,

0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25,

0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92,

0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84,

0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06,

0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b,

0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xe6,0x73,

0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e,

0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b,

0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0,0xfe,0x78,0xcd,0x5a,0xf4,

0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10,0x59,0x27,0x80,0xec,0x5f,

0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,0x2d,0xe5,0x7a,0x9f,0x93,0xc9,0x9c,0xef,

0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61,

0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14,0x63,0x55,0x21,0x0c,0x7d][self]

end

end

class Integer

def rcon # http://en.wikipedia.org/wiki/Rijndael\_key\_schedule#Rcon

[0x8d,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a][self]

end

end

puts 'Testing expand key:'

k="12345612345612345612345612345612".to\_bytes

puts k.expand\_key.pretty

def encrypt(m,k)

# Key expansion

k = k.expand\_key

# First Round

c = m.xor\_round\_key(k,0)

# Middle rounds

9.times {|i|

c = c.sub\_bytes

c = c.shift\_rows

c = c.mix\_cols

c = c.xor\_round\_key(k,i+1)

}

# Final Round

c = c.sub\_bytes

c = c.shift\_rows

c = c.xor\_round\_key(k,10)

end

class Array

def xor\_round\_key(keys,i)

xor(keys[i\*16,16])

end

end

class Array

def shift\_rows

[0,5,10,15,4,9,14,3,8,13,2,7,12,1,6,11].map{|i|self[i]}

end

def shift\_rows\_inv

[0,13,10,7,4,1,14,11,8,5,2,15,12,9,6,3].map{|i|self[i]}

end

end

class Array

def mix\_cols

self[0,4].mix\_col +

self[4,4].mix\_col +

self[8,4].mix\_col +

self[12,4].mix\_col

end

def mix\_cols\_inv

self[0,4].mix\_col\_inv +

self[4,4].mix\_col\_inv +

self[8,4].mix\_col\_inv +

self[12,4].mix\_col\_inv

end

end

class Array

def mix\_col # http://en.wikipedia.org/wiki/Rijndael\_mix\_columns

a0,a1,a2,a3 = self

[ a0.g2 ^ a1.g3 ^ a2 ^ a3, # r0 = 2a0 + 3a1 + 1a2 + 1a3

a0 ^ a1.g2 ^ a2.g3 ^ a3, # r1 = 1a0 + 2a1 + 3a2 + 1a3

a0 ^ a1 ^ a2.g2 ^ a3.g3, # r2 = 1a0 + 1a1 + 2a2 + 3a3

a0.g3 ^ a1 ^ a2 ^ a3.g2] # r3 = 3a0 + 1a1 + 1a2 + 2a3

end

end

class Integer

def g2 # Perform a galois field multiplication of the current value by 2

# Lookup table from here: http://code.google.com/p/byte-oriented-aes/

[0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,

0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,

0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,

0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,

0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,

0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,

0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,

0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,

0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,

0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,

0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,

0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,

0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,

0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,

0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,

0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5][self]

end

def g3

[0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,

0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,

0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,

0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,

0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,

0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,

0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,

0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,

0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,

0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,

0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xe9,0xea,

0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,

0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,

0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,

0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,

0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a][self]

end

end

puts 'Testing mix column:'

puts "db135345".to\_bytes.mix\_col.pretty # Should output: 8e4da1bc

puts "f20a225c".to\_bytes.mix\_col.pretty # Should output: 9fdc589d

puts "01010101".to\_bytes.mix\_col.pretty # Should output: 01010101

puts "c6c6c6c6".to\_bytes.mix\_col.pretty # Should output: c6c6c6c6

puts "d4d4d4d5".to\_bytes.mix\_col.pretty # Should output: d5d5d7d6

puts "2d26314c".to\_bytes.mix\_col.pretty # Should output: 4d7ebdf8

class Array

def mix\_col\_inv # http://en.wikipedia.org/wiki/Rijndael\_mix\_columns

a0,a1,a2,a3 = self

[ a0.g14 ^ a3.g9 ^ a2.g13 ^ a1.g11, # r0 = 14a0 + 9a3 + 13a2 + 11a1

a1.g14 ^ a0.g9 ^ a3.g13 ^ a2.g11, # r1 = 14a1 + 9a0 + 13a3 + 11a2

a2.g14 ^ a1.g9 ^ a0.g13 ^ a3.g11, # r2 = 14a2 + 9a1 + 13a0 + 11a3

a3.g14 ^ a2.g9 ^ a1.g13 ^ a0.g11] # r3 = 14a3 + 9a2 + 13a1 + 11a0

end

end

class Integer

def g9 # Galois multiply by 9

[0x00,0x09,0x12,0x1b,0x24,0x2d,0x36,0x3f,0x48,0x41,0x5a,0x53,0x6c,0x65,0x7e,0x77,

0x90,0x99,0x82,0x8b,0xb4,0xbd,0xa6,0xaf,0xd8,0xd1,0xca,0xc3,0xfc,0xf5,0xee,0xe7,

0x3b,0x32,0x29,0x20,0x1f,0x16,0x0d,0x04,0x73,0x7a,0x61,0x68,0x57,0x5e,0x45,0x4c,

0xab,0xa2,0xb9,0xb0,0x8f,0x86,0x9d,0x94,0xe3,0xea,0xf1,0xf8,0xc7,0xce,0xd5,0xdc,

0x76,0x7f,0x64,0x6d,0x52,0x5b,0x40,0x49,0x3e,0x37,0x2c,0x25,0x1a,0x13,0x08,0x01,

0xe6,0xef,0xf4,0xfd,0xc2,0xcb,0xd0,0xd9,0xae,0xa7,0xbc,0xb5,0x8a,0x83,0x98,0x91,

0x4d,0x44,0x5f,0x56,0x69,0x60,0x7b,0x72,0x05,0x0c,0x17,0x1e,0x21,0x28,0x33,0x3a,

0xdd,0xd4,0xcf,0xc6,0xf9,0xf0,0xeb,0xe2,0x95,0x9c,0x87,0x8e,0xb1,0xb8,0xa3,0xaa,

0xec,0xe5,0xfe,0xf7,0xc8,0xc1,0xda,0xd3,0xa4,0xad,0xb6,0xbf,0x80,0x89,0x92,0x9b,

0x7c,0x75,0x6e,0x67,0x58,0x51,0x4a,0x43,0x34,0x3d,0x26,0x2f,0x10,0x19,0x02,0x0b,

0xd7,0xde,0xc5,0xcc,0xf3,0xfa,0xe1,0xe8,0x9f,0x96,0x8d,0x84,0xbb,0xb2,0xa9,0xa0,

0x47,0x4e,0x55,0x5c,0x63,0x6a,0x71,0x78,0x0f,0x06,0x1d,0x14,0x2b,0x22,0x39,0x30,

0x9a,0x93,0x88,0x81,0xbe,0xb7,0xac,0xa5,0xd2,0xdb,0xc0,0xc9,0xf6,0xff,0xe4,0xed,

0x0a,0x03,0x18,0x11,0x2e,0x27,0x3c,0x35,0x42,0x4b,0x50,0x59,0x66,0x6f,0x74,0x7d,

0xa1,0xa8,0xb3,0xba,0x85,0x8c,0x97,0x9e,0xe9,0xe0,0xfb,0xf2,0xcd,0xc4,0xdf,0xd6,

0x31,0x38,0x23,0x2a,0x15,0x1c,0x07,0x0e,0x79,0x70,0x6b,0x62,0x5d,0x54,0x4f,0x46][self]

end

def g11

[0x00,0x0b,0x16,0x1d,0x2c,0x27,0x3a,0x31,0x58,0x53,0x4e,0x45,0x74,0x7f,0x62,0x69,

0xb0,0xbb,0xa6,0xad,0x9c,0x97,0x8a,0x81,0xe8,0xe3,0xfe,0xf5,0xc4,0xcf,0xd2,0xd9,

0x7b,0x70,0x6d,0x66,0x57,0x5c,0x41,0x4a,0x23,0x28,0x35,0x3e,0x0f,0x04,0x19,0x12,

0xcb,0xc0,0xdd,0xd6,0xe7,0xec,0xf1,0xfa,0x93,0x98,0x85,0x8e,0xbf,0xb4,0xa9,0xa2,

0xf6,0xfd,0xe0,0xeb,0xda,0xd1,0xcc,0xc7,0xae,0xa5,0xb8,0xb3,0x82,0x89,0x94,0x9f,

0x46,0x4d,0x50,0x5b,0x6a,0x61,0x7c,0x77,0x1e,0x15,0x08,0x03,0x32,0x39,0x24,0x2f,

0x8d,0x86,0x9b,0x90,0xa1,0xaa,0xb7,0xbc,0xd5,0xde,0xc3,0xc8,0xf9,0xf2,0xef,0xe4,

0x3d,0x36,0x2b,0x20,0x11,0x1a,0x07,0x0c,0x65,0x6e,0x73,0x78,0x49,0x42,0x5f,0x54,

0xf7,0xfc,0xe1,0xea,0xdb,0xd0,0xcd,0xc6,0xaf,0xa4,0xb9,0xb2,0x83,0x88,0x95,0x9e,

0x47,0x4c,0x51,0x5a,0x6b,0x60,0x7d,0x76,0x1f,0x14,0x09,0x02,0x33,0x38,0x25,0x2e,

0x8c,0x87,0x9a,0x91,0xa0,0xab,0xb6,0xbd,0xd4,0xdf,0xc2,0xc9,0xf8,0xf3,0xee,0xe5,

0x3c,0x37,0x2a,0x21,0x10,0x1b,0x06,0x0d,0x64,0x6f,0x72,0x79,0x48,0x43,0x5e,0x55,

0x01,0x0a,0x17,0x1c,0x2d,0x26,0x3b,0x30,0x59,0x52,0x4f,0x44,0x75,0x7e,0x63,0x68,

0xb1,0xba,0xa7,0xac,0x9d,0x96,0x8b,0x80,0xe9,0xe2,0xff,0xf4,0xc5,0xce,0xd3,0xd8,

0x7a,0x71,0x6c,0x67,0x56,0x5d,0x40,0x4b,0x22,0x29,0x34,0x3f,0x0e,0x05,0x18,0x13,

0xca,0xc1,0xdc,0xd7,0xe6,0xed,0xf0,0xfb,0x92,0x99,0x84,0x8f,0xbe,0xb5,0xa8,0xa3][self]

end

def g13

[0x00,0x0d,0x1a,0x17,0x34,0x39,0x2e,0x23,0x68,0x65,0x72,0x7f,0x5c,0x51,0x46,0x4b,

0xd0,0xdd,0xca,0xc7,0xe4,0xe9,0xfe,0xf3,0xb8,0xb5,0xa2,0xaf,0x8c,0x81,0x96,0x9b,

0xbb,0xb6,0xa1,0xac,0x8f,0x82,0x95,0x98,0xd3,0xde,0xc9,0xc4,0xe7,0xea,0xfd,0xf0,

0x6b,0x66,0x71,0x7c,0x5f,0x52,0x45,0x48,0x03,0x0e,0x19,0x14,0x37,0x3a,0x2d,0x20,

0x6d,0x60,0x77,0x7a,0x59,0x54,0x43,0x4e,0x05,0x08,0x1f,0x12,0x31,0x3c,0x2b,0x26,

0xbd,0xb0,0xa7,0xaa,0x89,0x84,0x93,0x9e,0xd5,0xd8,0xcf,0xc2,0xe1,0xec,0xfb,0xf6,

0xd6,0xdb,0xcc,0xc1,0xe2,0xef,0xf8,0xf5,0xbe,0xb3,0xa4,0xa9,0x8a,0x87,0x90,0x9d,

0x06,0x0b,0x1c,0x11,0x32,0x3f,0x28,0x25,0x6e,0x63,0x74,0x79,0x5a,0x57,0x40,0x4d,

0xda,0xd7,0xc0,0xcd,0xee,0xe3,0xf4,0xf9,0xb2,0xbf,0xa8,0xa5,0x86,0x8b,0x9c,0x91,

0x0a,0x07,0x10,0x1d,0x3e,0x33,0x24,0x29,0x62,0x6f,0x78,0x75,0x56,0x5b,0x4c,0x41,

0x61,0x6c,0x7b,0x76,0x55,0x58,0x4f,0x42,0x09,0x04,0x13,0x1e,0x3d,0x30,0x27,0x2a,

0xb1,0xbc,0xab,0xa6,0x85,0x88,0x9f,0x92,0xd9,0xd4,0xc3,0xce,0xed,0xe0,0xf7,0xfa,

0xb7,0xba,0xad,0xa0,0x83,0x8e,0x99,0x94,0xdf,0xd2,0xc5,0xc8,0xeb,0xe6,0xf1,0xfc,

0x67,0x6a,0x7d,0x70,0x53,0x5e,0x49,0x44,0x0f,0x02,0x15,0x18,0x3b,0x36,0x21,0x2c,

0x0c,0x01,0x16,0x1b,0x38,0x35,0x22,0x2f,0x64,0x69,0x7e,0x73,0x50,0x5d,0x4a,0x47,

0xdc,0xd1,0xc6,0xcb,0xe8,0xe5,0xf2,0xff,0xb4,0xb9,0xae,0xa3,0x80,0x8d,0x9a,0x97][self]

end

def g14

[0x00,0x0e,0x1c,0x12,0x38,0x36,0x24,0x2a,0x70,0x7e,0x6c,0x62,0x48,0x46,0x54,0x5a,

0xe0,0xee,0xfc,0xf2,0xd8,0xd6,0xc4,0xca,0x90,0x9e,0x8c,0x82,0xa8,0xa6,0xb4,0xba,

0xdb,0xd5,0xc7,0xc9,0xe3,0xed,0xff,0xf1,0xab,0xa5,0xb7,0xb9,0x93,0x9d,0x8f,0x81,

0x3b,0x35,0x27,0x29,0x03,0x0d,0x1f,0x11,0x4b,0x45,0x57,0x59,0x73,0x7d,0x6f,0x61,

0xad,0xa3,0xb1,0xbf,0x95,0x9b,0x89,0x87,0xdd,0xd3,0xc1,0xcf,0xe5,0xeb,0xf9,0xf7,

0x4d,0x43,0x51,0x5f,0x75,0x7b,0x69,0x67,0x3d,0x33,0x21,0x2f,0x05,0x0b,0x19,0x17,

0x76,0x78,0x6a,0x64,0x4e,0x40,0x52,0x5c,0x06,0x08,0x1a,0x14,0x3e,0x30,0x22,0x2c,

0x96,0x98,0x8a,0x84,0xae,0xa0,0xb2,0xbc,0xe6,0xe8,0xfa,0xf4,0xde,0xd0,0xc2,0xcc,

0x41,0x4f,0x5d,0x53,0x79,0x77,0x65,0x6b,0x31,0x3f,0x2d,0x23,0x09,0x07,0x15,0x1b,

0xa1,0xaf,0xbd,0xb3,0x99,0x97,0x85,0x8b,0xd1,0xdf,0xcd,0xc3,0xe9,0xe7,0xf5,0xfb,

0x9a,0x94,0x86,0x88,0xa2,0xac,0xbe,0xb0,0xea,0xe4,0xf6,0xf8,0xd2,0xdc,0xce,0xc0,

0x7a,0x74,0x66,0x68,0x42,0x4c,0x5e,0x50,0x0a,0x04,0x16,0x18,0x32,0x3c,0x2e,0x20,

0xec,0xe2,0xf0,0xfe,0xd4,0xda,0xc8,0xc6,0x9c,0x92,0x80,0x8e,0xa4,0xaa,0xb8,0xb6,

0x0c,0x02,0x10,0x1e,0x34,0x3a,0x28,0x26,0x7c,0x72,0x60,0x6e,0x44,0x4a,0x58,0x56,

0x37,0x39,0x2b,0x25,0x0f,0x01,0x13,0x1d,0x47,0x49,0x5b,0x55,0x7f,0x71,0x63,0x6d,

0xd7,0xd9,0xcb,0xc5,0xef,0xe1,0xf3,0xfd,0xa7,0xa9,0xbb,0xb5,0x9f,0x91,0x83,0x8d][self]

end

end

puts 'Testing mix column inverse:'

puts "8e4da1bc".to\_bytes.mix\_col\_inv.pretty # Should output: db135345

puts "9fdc589d".to\_bytes.mix\_col\_inv.pretty # Should output: f20a225c

puts "01010101".to\_bytes.mix\_col\_inv.pretty # Should output: 01010101

puts "c6c6c6c6".to\_bytes.mix\_col\_inv.pretty # Should output: c6c6c6c6

puts "d5d5d7d6".to\_bytes.mix\_col\_inv.pretty # Should output: d4d4d4d5

puts "4d7ebdf8".to\_bytes.mix\_col\_inv.pretty # Should output: 2d26314c

puts 'Testing encryption:'

m="abcdefabcdefabcdefabcdefabcdefab".to\_bytes

k="12345612345612345612345612345612".to\_bytes

c=encrypt(m,k)

puts c.pretty # Should be 85E5A3D7356A61E29A8AFA559AD67102

def decrypt(c,k)

# Key expansion

k = k.expand\_key

# Reverse the final round

c = c.xor\_round\_key(k,10)

c = c.shift\_rows\_inv

c = c.sub\_bytes\_inv

# Reverse the middle rounds

9.times {|i|

c = c.xor\_round\_key(k,9-i)

c = c.mix\_cols\_inv

c = c.shift\_rows\_inv

c = c.sub\_bytes\_inv

}

# Reverse the first round

m = c.xor\_round\_key(k,0)

end

puts 'Testing decryption:'

c="85E5A3D7356A61E29A8AFA559AD67102".to\_bytes

k="12345612345612345612345612345612".to\_bytes

m=decrypt(c,k)

puts m.pretty # Should be abcdefabcdefabcdefabcdefabcdefab