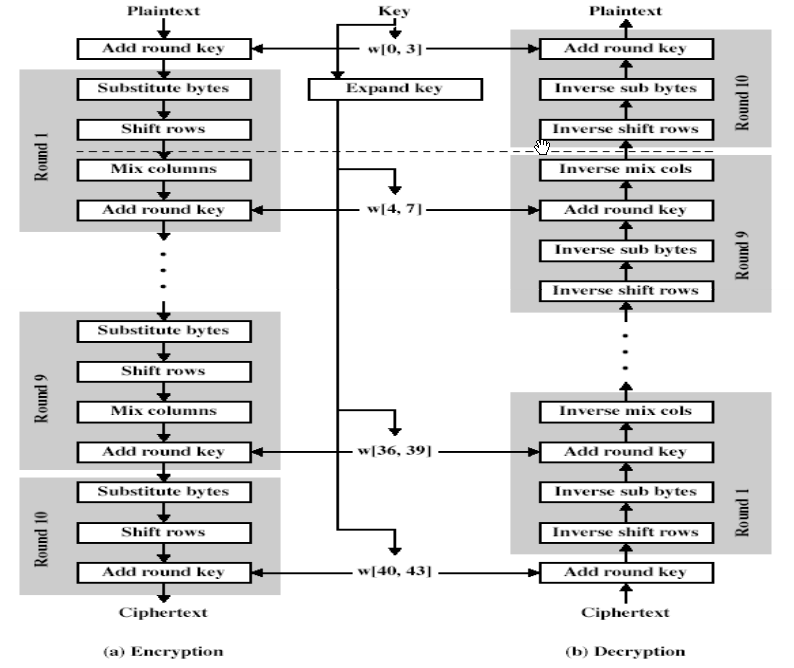
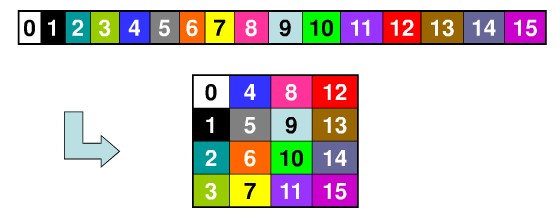
先搞定AES算法，基本变换包括SubBytes（字节替代）、ShiftRows（行移位）、MixColumns（列混淆）、AddRoundKey(轮密钥加）

其算法一般描述为



明文及密钥的组织排列方式



ByteSubstitution（字节替代）

非线性的字节替代，单独处理每个字节：

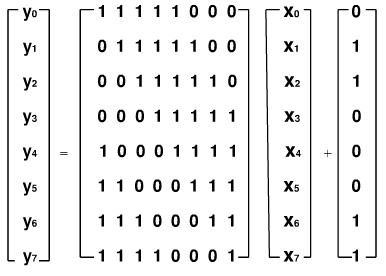
求该字节在有限域GF(28)上的乘法逆，"0"被映射为自身，即对于α∈GF(28)，求β∈GF(28)，

使得α·β=β·α=1mod(x8+x4+x2+x+1)。

对上一步求得的乘法逆作仿射变换

yi=xi + x(i+4)mod8 + x(i+6)mod8 + x(i+7)mod8 + ci

(其中ci是6310即011000112的第i位），用矩阵表示为



本来打算把求乘法逆和仿射变换算法敲上去，最后还是放弃了...直接打置换表

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | unsigned char sBox[] =  { /\*  0    1    2    3    4    5    6    7    8    9    a    b    c    d    e    f \*/      0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76, /\*0\*/      0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0, /\*1\*/      0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15, /\*2\*/      0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75, /\*3\*/      0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84, /\*4\*/      0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf, /\*5\*/      0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8, /\*6\*/      0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2, /\*7\*/      0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73, /\*8\*/      0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb, /\*9\*/      0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79, /\*a\*/      0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08, /\*b\*/      0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a, /\*c\*/      0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e, /\*d\*/      0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf, /\*e\*/      0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16  /\*f\*/  }; |

 下面是逆置换表，解密时使用

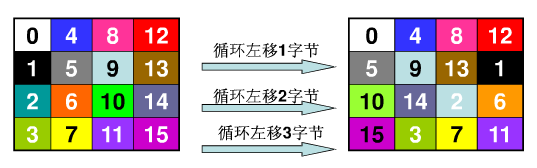
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | unsigned char invsBox[256] =  { /\*  0    1    2    3    4    5    6    7    8    9    a    b    c    d    e    f  \*/      0x52,0x09,0x6a,0xd5,0x30,0x36,0xa5,0x38,0xbf,0x40,0xa3,0x9e,0x81,0xf3,0xd7,0xfb, /\*0\*/      0x7c,0xe3,0x39,0x82,0x9b,0x2f,0xff,0x87,0x34,0x8e,0x43,0x44,0xc4,0xde,0xe9,0xcb, /\*1\*/      0x54,0x7b,0x94,0x32,0xa6,0xc2,0x23,0x3d,0xee,0x4c,0x95,0x0b,0x42,0xfa,0xc3,0x4e, /\*2\*/      0x08,0x2e,0xa1,0x66,0x28,0xd9,0x24,0xb2,0x76,0x5b,0xa2,0x49,0x6d,0x8b,0xd1,0x25, /\*3\*/      0x72,0xf8,0xf6,0x64,0x86,0x68,0x98,0x16,0xd4,0xa4,0x5c,0xcc,0x5d,0x65,0xb6,0x92, /\*4\*/      0x6c,0x70,0x48,0x50,0xfd,0xed,0xb9,0xda,0x5e,0x15,0x46,0x57,0xa7,0x8d,0x9d,0x84, /\*5\*/      0x90,0xd8,0xab,0x00,0x8c,0xbc,0xd3,0x0a,0xf7,0xe4,0x58,0x05,0xb8,0xb3,0x45,0x06, /\*6\*/      0xd0,0x2c,0x1e,0x8f,0xca,0x3f,0x0f,0x02,0xc1,0xaf,0xbd,0x03,0x01,0x13,0x8a,0x6b, /\*7\*/      0x3a,0x91,0x11,0x41,0x4f,0x67,0xdc,0xea,0x97,0xf2,0xcf,0xce,0xf0,0xb4,0xe6,0x73, /\*8\*/      0x96,0xac,0x74,0x22,0xe7,0xad,0x35,0x85,0xe2,0xf9,0x37,0xe8,0x1c,0x75,0xdf,0x6e, /\*9\*/      0x47,0xf1,0x1a,0x71,0x1d,0x29,0xc5,0x89,0x6f,0xb7,0x62,0x0e,0xaa,0x18,0xbe,0x1b, /\*a\*/      0xfc,0x56,0x3e,0x4b,0xc6,0xd2,0x79,0x20,0x9a,0xdb,0xc0,0xfe,0x78,0xcd,0x5a,0xf4, /\*b\*/      0x1f,0xdd,0xa8,0x33,0x88,0x07,0xc7,0x31,0xb1,0x12,0x10,0x59,0x27,0x80,0xec,0x5f, /\*c\*/      0x60,0x51,0x7f,0xa9,0x19,0xb5,0x4a,0x0d,0x2d,0xe5,0x7a,0x9f,0x93,0xc9,0x9c,0xef, /\*d\*/      0xa0,0xe0,0x3b,0x4d,0xae,0x2a,0xf5,0xb0,0xc8,0xeb,0xbb,0x3c,0x83,0x53,0x99,0x61, /\*e\*/      0x17,0x2b,0x04,0x7e,0xba,0x77,0xd6,0x26,0xe1,0x69,0x14,0x63,0x55,0x21,0x0c,0x7d  /\*f\*/  }; |

这里遇到问题了，本来用纯c初始化数组很正常，封装成类以后发现不能初始化，不管是声明、构造函数都无法初始化，百歌谷度了一通后没有任何答案，无奈只能在构造函数中声明一个局部变量数组并初始化，然后用memcpy，（成员变量名为Sbox/InvSbox，局部变量名sBox/invsBox）

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | void AES::SubBytes(unsigned char state[][4])  {      int r,c;      for(r=0; r<4; r++)      {          for(c=0; c<4; c++)          {              state[r][c] = Sbox[state[r][c]];          }      }  } |

 ShiftRows（行移位变换）

行移位变换完成基于行的循环位移操作，变换方法：



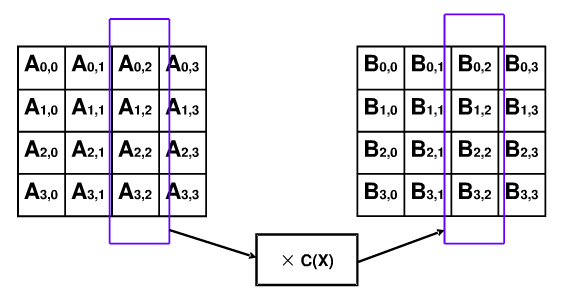
即行移位变换作用于行上，第0行不变，第1行循环左移1个字节，第2行循环左移2个字节，第3行循环左移3个字节。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | void AES::ShiftRows(unsigned char state[][4])  {      unsigned char t[4];      int r,c;      for(r=1; r<4; r++)      {          for(c=0; c<4; c++)          {              t[c] = state[r][(c+r)%4];          }          for(c=0; c<4; c++)          {              state[r][c] = t[c];          }      }  } |

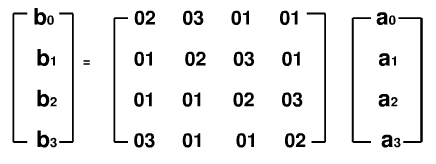
MixColumns（列混淆变换）

 逐列混合，方法：

b(x) = (03·x3 + 01·x2 + 01·x + 02) · a(x) mod(x4 + 1)



矩阵表示形式：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | void AES::MixColumns(unsigned char state[][4])  {      unsigned char t[4];      int r,c;      for(c=0; c< 4; c++)      {          for(r=0; r<4; r++)          {              t[r] = state[r][c];          }          for(r=0; r<4; r++)          {              state[r][c] = FFmul(0x02, t[r])                          ^ FFmul(0x03, t[(r+1)%4])                          ^ FFmul(0x01, t[(r+2)%4])                          ^ FFmul(0x01, t[(r+3)%4]);          }      }  }    unsigned char AES::FFmul(unsigned char a, unsigned char b)  {      unsigned char bw[4];      unsigned char res=0;      int i;      bw[0] = b;      for(i=1; i<4; i++)      {          bw[i] = bw[i-1]<<1;          if(bw[i-1]&0x80)          {              bw[i]^=0x1b;          }      }      for(i=0; i<4; i++)      {          if((a>>i)&0x01)          {              res ^= bw[i];          }      }      return res;  } |

其中FFmul为有限域GF(28)上的乘法，标准算法应该是循环8次（b与a的每一位相乘，结果相加），但这里只用到最低2位，解密时用到的逆列混淆也只用了低4位，所以在这里高4位的运算是多余的，只计算低4位。

AddRoundKey（轮密钥加变换）

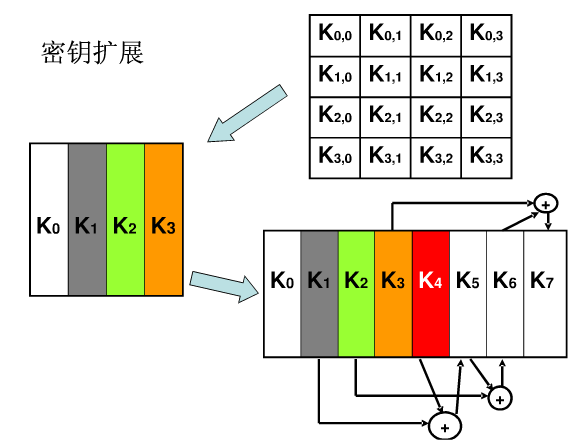
 简单来说就是逐字节相加，有限域GF(28)上的加法是模2加法，即异或

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | void AES::AddRoundKey(unsigned char state[][4], unsigned char k[][4])  {      int r,c;      for(c=0; c<4; c++)      {          for(r=0; r<4; r++)          {              state[r][c] ^= k[r][c];          }      }  } |

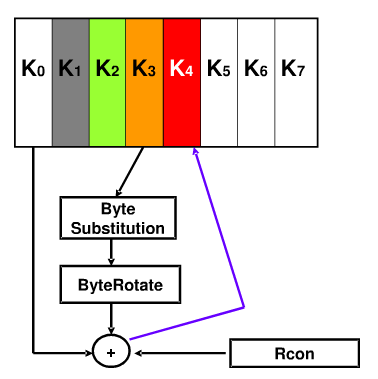
**KeyExpansion（密钥扩展）**

 将输入的密钥扩展为11组128位密钥组，其中第0组为输入密钥本身

 其后第n组第i列 为 第n-1组第i列 与 第n组第i-1列之和（模2加法，1<= i <=3）



对于每一组 第一列即i=0，有特殊的处理



将前一列即第n-1组第3列的4个字节循环左移1个字节，

并对每个字节进行字节替代变换SubBytes

将第一行（即第一个字节）与轮常量rc[n]相加

最后再与前一组该列相加

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | void AES::KeyExpansion(unsigned char\* key, unsigned char w[][4][4])  {      int i,j,r,c;      unsigned char rc[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36};      for(r=0; r<4; r++)      {          for(c=0; c<4; c++)          {              w[0][r][c] = key[r+c\*4];          }      }      for(i=1; i<=10; i++)      {          for(j=0; j<4; j++)          {              unsigned char t[4];              for(r=0; r<4; r++)              {                  t[r] = j ? w[i][r][j-1] : w[i-1][r][3];              }              if(j == 0)              {                  unsigned char temp = t[0];                  for(r=0; r<3; r++)                  {                      t[r] = Sbox[t[(r+1)%4]];                  }                  t[3] = Sbox[temp];                  t[0] ^= rc[i-1];              }              for(r=0; r<4; r++)              {                  w[i][r][j] = w[i-1][r][j] ^ t[r];              }          }      }  } |

**解密的基本运算**

AES解密算法与加密不同，基本运算中除了AddRoundKey（轮密钥加）不变外，其余的都需要进行逆变换，即

InvSubBytes（逆字节替代）、InvShiftRows（逆行移位）、InvMixColumns（逆列混淆）

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | void AES::InvSubBytes(unsigned char state[][4])  {      int r,c;      for(r=0; r<4; r++)      {          for(c=0; c<4; c++)          {              state[r][c] = InvSbox[state[r][c]];          }      }  }    void AES::InvShiftRows(unsigned char state[][4])  {      unsigned char t[4];      int r,c;      for(r=1; r<4; r++)      {          for(c=0; c<4; c++)          {              t[c] = state[r][(c-r+4)%4];          }          for(c=0; c<4; c++)          {              state[r][c] = t[c];          }      }  }    void AES::InvMixColumns(unsigned char state[][4])  {      unsigned char t[4];      int r,c;      for(c=0; c< 4; c++)      {          for(r=0; r<4; r++)          {              t[r] = state[r][c];          }          for(r=0; r<4; r++)          {              state[r][c] = FFmul(0x0e, t[r])                          ^ FFmul(0x0b, t[(r+1)%4])                          ^ FFmul(0x0d, t[(r+2)%4])                          ^ FFmul(0x09, t[(r+3)%4]);          }      }  } |

**加密过程**

先将输入的明文按列序组合成4\*4的矩阵，直接与第0组密钥（即输入的密钥）相加（异或），作为轮加密的输入

然后循环10次进行SubBytes、ShiftRows、MixColumns、AddRoundKey运算，最后恢复原序列

需要注意的是最后一轮并不进行MixColumns（列混淆变换）

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | unsigned char\* AES::Cipher(unsigned char\* input)  {      unsigned char state[4][4];      int i,r,c;        for(r=0; r<4; r++)      {          for(c=0; c<4 ;c++)          {              state[r][c] = input[c\*4+r];          }      }        AddRoundKey(state,w[0]);        for(i=1; i<=10; i++)      {          SubBytes(state);          ShiftRows(state);          if(i!=10)MixColumns(state);          AddRoundKey(state,w[i]);      }        for(r=0; r<4; r++)      {          for(c=0; c<4 ;c++)          {              input[c\*4+r] = state[r][c];          }      }        return input;  } |

**解密过程**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | unsigned char\* AES::InvCipher(unsigned char\* input)  {      unsigned char state[4][4];      int i,r,c;        for(r=0; r<4; r++)      {          for(c=0; c<4 ;c++)          {              state[r][c] = input[c\*4+r];          }      }        AddRoundKey(state, w[10]);      for(i=9; i>=0; i--)      {          InvShiftRows(state);          InvSubBytes(state);          AddRoundKey(state, w[i]);          if(i)InvMixColumns(state);      }        for(r=0; r<4; r++)      {          for(c=0; c<4 ;c++)          {              input[c\*4+r] = state[r][c];          }      }        return input;  } |

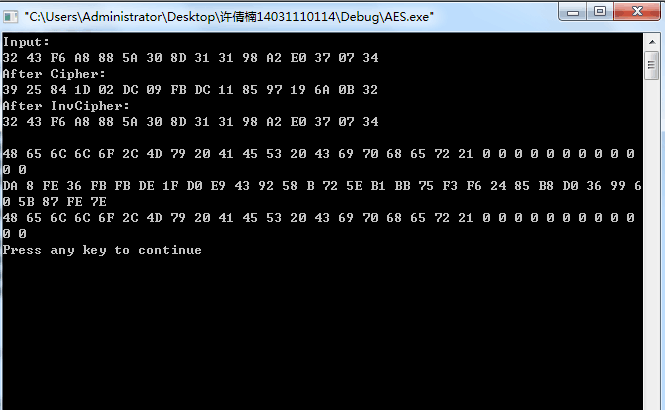
**对外部数据的加密/解密**

至此已经实现了AES加密与解密的原型，在使用的时候一般处理的是字符串等，而不是直接传入128位的数据，所以要封装一下对外部数据的加解密处理

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | void\* AES::Cipher(void\* input, int length)  {      unsigned char\* in = (unsigned char\*) input;      int i;      if(!length)      {          while(\*(in+length++));          in = (unsigned char\*) input;      }      for(i=0; i<length; i+=16)      {          Cipher(in+i);      }      return input;  }    void\* AES::InvCipher(void\* input, int length)  {      unsigned char\* in = (unsigned char\*) input;      int i;      for(i=0; i<length; i+=16)      {          InvCipher(in+i);      }      return input;  } |

加密时默认参数length=0,为要加密的数据长度，如果使用默认值，则作为字符串处理，以'\0'为结尾计算长度

加密时传进的指针要预留够16整数倍字节的空间，因为加密操作直接修改原数据，不足128位可能造成内存溢出



///////////////////////////////

// http://mingcn.cnblogs.com //

// xelz CopyRight (c) 2010 //

///////////////////////////////

#include "stdafx.h"

#include "string.h"

#include "AES.h"

memcpy(Sbox, sBox, 256);

memcpy(InvSbox, invsBox, 256);

KeyExpansion(key, w);

}

AES::~AES()

{

}

unsigned char\* AES::Cipher(unsigned char\* input)

{

unsigned char state[4][4];

int i,r,c;

for(r=0; r<4; r++)

{

for(c=0; c<4 ;c++)

{

state[r][c] = input[c\*4+r];

}

}

AddRoundKey(state,w[0]);

for(i=1; i<=10; i++)

{

SubBytes(state);

ShiftRows(state);

if(i!=10)MixColumns(state);

AddRoundKey(state,w[i]);

}

for(r=0; r<4; r++)

{

for(c=0; c<4 ;c++)

{

input[c\*4+r] = state[r][c];

}

}

return input;

}

unsigned char\* AES::InvCipher(unsigned char\* input)

{

unsigned char state[4][4];

int i,r,c;

for(r=0; r<4; r++)

{

for(c=0; c<4 ;c++)

{

state[r][c] = input[c\*4+r];

}

}

AddRoundKey(state, w[10]);

for(i=9; i>=0; i--)

{

InvShiftRows(state);

InvSubBytes(state);

AddRoundKey(state, w[i]);

if(i)

{

InvMixColumns(state);

}

}

for(r=0; r<4; r++)

{

for(c=0; c<4 ;c++)

{

input[c\*4+r] = state[r][c];

}

}

return input;

}

void\* AES::Cipher(void\* input, int length)

{

unsigned char\* in = (unsigned char\*) input;

int i;

if(!length)

{

while(\*(in+length++));

in = (unsigned char\*) input;

}

for(i=0; i<length; i+=16)

{

Cipher(in+i);

}

return input;

}

void\* AES::InvCipher(void\* input, int length)

{

unsigned char\* in = (unsigned char\*) input;

int i;

for(i=0; i<length; i+=16)

{

InvCipher(in+i);

}

return input;

}

void AES::KeyExpansion(unsigned char\* key, unsigned char w[][4][4])

{

int i,j,r,c;

unsigned char rc[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36};

for(r=0; r<4; r++)

{

for(c=0; c<4; c++)

{

w[0][r][c] = key[r+c\*4];

}

}

for(i=1; i<=10; i++)

{

for(j=0; j<4; j++)

{

unsigned char t[4];

for(r=0; r<4; r++)

{

t[r] = j ? w[i][r][j-1] : w[i-1][r][3];

}

if(j == 0)

{

unsigned char temp = t[0];

for(r=0; r<3; r++)

{

t[r] = Sbox[t[(r+1)%4]];

}

t[3] = Sbox[temp];

t[0] ^= rc[i-1];

}

for(r=0; r<4; r++)

{

w[i][r][j] = w[i-1][r][j] ^ t[r];

}

}

}

}

unsigned char AES::FFmul(unsigned char a, unsigned char b)

{

unsigned char bw[4];

unsigned char res=0;

int i;

bw[0] = b;

for(i=1; i<4; i++)

{

bw[i] = bw[i-1]<<1;

if(bw[i-1]&0x80)

{

bw[i]^=0x1b;

}

}

for(i=0; i<4; i++)

{

if((a>>i)&0x01)

{

res ^= bw[i];

}

}

return res;

}

void AES::SubBytes(unsigned char state[][4])

{

int r,c;

for(r=0; r<4; r++)

{

for(c=0; c<4; c++)

{

state[r][c] = Sbox[state[r][c]];

}

}

}

void AES::ShiftRows(unsigned char state[][4])

{

unsigned char t[4];

int r,c;

for(r=1; r<4; r++)

{

for(c=0; c<4; c++)

{

t[c] = state[r][(c+r)%4];

}

for(c=0; c<4; c++)

{

state[r][c] = t[c];

}

}

}

void AES::MixColumns(unsigned char state[][4])

{

unsigned char t[4];

int r,c;

for(c=0; c< 4; c++)

{

for(r=0; r<4; r++)

{

t[r] = state[r][c];

}

for(r=0; r<4; r++)

{

state[r][c] = FFmul(0x02, t[r])

^ FFmul(0x03, t[(r+1)%4])

^ FFmul(0x01, t[(r+2)%4])

^ FFmul(0x01, t[(r+3)%4]);

}

}

}

void AES::AddRoundKey(unsigned char state[][4], unsigned char k[][4])

{

int r,c;

for(c=0; c<4; c++)

{

for(r=0; r<4; r++)

{

state[r][c] ^= k[r][c];

}

}

}

void AES::InvSubBytes(unsigned char state[][4])

{

int r,c;

for(r=0; r<4; r++)

{

for(c=0; c<4; c++)

{

state[r][c] = InvSbox[state[r][c]];

}

}

}

void AES::InvShiftRows(unsigned char state[][4])

{

unsigned char t[4];

int r,c;

for(r=1; r<4; r++)

{

for(c=0; c<4; c++)

{

t[c] = state[r][(c-r+4)%4];

}

for(c=0; c<4; c++)

{

state[r][c] = t[c];

}

}

}

void AES::InvMixColumns(unsigned char state[][4])

{

unsigned char t[4];

int r,c;

for(c=0; c< 4; c++)

{

for(r=0; r<4; r++)

{

t[r] = state[r][c];

}

for(r=0; r<4; r++)

{

state[r][c] = FFmul(0x0e, t[r])

^ FFmul(0x0b, t[(r+1)%4])

^ FFmul(0x0d, t[(r+2)%4])

^ FFmul(0x09, t[(r+3)%4]);

}

}

}