/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FileName:

SM4.cpp Version:

SM4\_1.0 Date:

Sep 13,2016 Description:

This code provide the implement of SM4 algorithm,which has the bolck length of 16 bytes and key length of 16 bytes.

SM4 algorithm consists of 32 rounds,thus it generate 32 round keys which has a length of

16 bytes.

Function List:

1. SM4\_KeySchedule //Generate the required round keys

2. SM4\_Encrypt //Encryption fuction

3. SM4\_Decrypt //Decryption fuction History:

Date:Sep 13,2016

Author:Mao Yingying,Huo Lili

Modification: 1)add notes to all the functions 2)add SM4\_SelfCheck function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "SM4.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function:

void SM4\_KeySchedule(unsigned char MK[], unsigned int rk[]);

Description:

Generate round keys Calls:

Called By:

SM4\_Encrypt;

SM4\_Decrypt; Input:

MK[]: Master key Output:

rk[]: round keys

Return:null Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SM4\_KeySchedule(unsigned char MK[],unsigned int rk[]) {

unsigned int tmp,buf,K[36]; int i;

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

K[i]=SM4\_FK[i]^( (MK[4\*i]<<24) | (MK[4\*i+1]<<16) | (MK[4\*i+2]<<8) | (MK[4\*i+3]) );

}

for(i=0;i<32;i++) {

tmp =K[i+1]^K[i+2]^K[i+3]^ SM4\_CK[i];

//nonlinear operation

buf= (SM4\_Sbox[(tmp >> 24) & 0xFF]) << 24 | (SM4\_Sbox[(tmp >> 16) & 0xFF]) << 16 | (SM4\_Sbox[(tmp >> 8) & 0xFF]) << 8 | (SM4\_Sbox[tmp & 0xFF]);

//linear operation

K[i+4]=K[i]^((buf)^(SM4\_Rotl32((buf),13))^(SM4\_Rotl32((buf),23)));

rk[i]=K[i+4]; }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function:

void SM4\_Encrypt(unsigned char MK[],unsigned char PlainText[],unsigned char CipherText[]);

Description:

Encryption function Calls:

SM4\_KeySchedule

Called By: Input:

MK[]: Master key

PlainText[]: input text Output:

CipherText[]: output text Return:null

Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SM4\_Encrypt(unsigned char MK[],unsigned char PlainText[],unsigned char CipherText[]) {

unsigned int rk[32],X[36],tmp,buf; int i,j;

SM4\_KeySchedule(MK,rk);

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

X[j]=(PlainText[j\*4]<<24) | (PlainText[j\*4+1]<<16) | (PlainText[j\*4+2]<<8) | (PlainText[j\*4+3]);

}

for(i=0;i<32;i++) {

tmp = X[i+1]^X[i+2]^X[i+3]^rk[i];

//nonlinear operation

buf= ( SM4\_Sbox[(tmp >> 24) & 0xFF]) << 24 | (SM4\_Sbox[(tmp >> 16) & 0xFF]) << 16 | (SM4\_Sbox[(tmp >> 8) & 0xFF]) << 8 | (SM4\_Sbox[tmp & 0xFF]);

//linear operation

X[i+4]=X[i]^(buf^SM4\_Rotl32((buf),2)^ SM4\_Rotl32((buf),10) ^ SM4\_Rotl32((buf),18)^ SM4\_Rotl32((buf),24));

}

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

CipherText[4\*j]=(X[35-j]>> 24)& 0xFF;

CipherText[4\*j+1]=(X[35-j]>> 16)& 0xFF; CipherText[4\*j+2]=(X[35-j]>> 8)& 0xFF; CipherText[4\*j+3]=(X[35-j])& 0xFF;

} }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function:

void SM4\_Decrypt(unsigned char MK[],unsigned char CipherText[], unsigned char PlainText[]); Description:

Decryption function Calls:

SM4\_KeySchedule Called By:

Input:

MK[]: Master key

CipherText[]: input text Output:

PlainText[]: output text

Return:null Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SM4\_Decrypt(unsigned char MK[],unsigned char CipherText[],unsigned char PlainText[]) {

unsigned int rk[32],X[36],tmp,buf; int i,j;

SM4\_KeySchedule(MK,rk);

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

X[j]=(CipherText[j\*4]<<24) | (CipherText[j\*4+1]<<16) | (CipherText[j\*4+2]<<8) | (CipherText[j\*4+3]);

}

for(i=0;i<32;i++) {

tmp = X[i+1]^X[i+2]^X[i+3]^rk[31-i];

//nonlinear operation

buf= (SM4\_Sbox[(tmp >> 24) & 0xFF]) << 24 | (SM4\_Sbox[(tmp >> 16) & 0xFF]) << 16 | (SM4\_Sbox[(tmp >> 8) & 0xFF]) << 8 | (SM4\_Sbox[tmp & 0xFF]);

//linear operation

X[i+4]=X[i]^(buf^SM4\_Rotl32((buf),2)^ SM4\_Rotl32((buf),10) ^ SM4\_Rotl32((buf),18)^ SM4\_Rotl32((buf),24));

}

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

PlainText[4\*j]=(X[35-j]>> 24)& 0xFF; PlainText[4\*j+1]=(X[35-j]>>16)& 0xFF; PlainText[4\*j+2]=(X[35-j]>> 8)& 0xFF;

PlainText[4\*j+3]=(X[35-j])& 0xFF; }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function:

int SM4\_SelfCheck() Description:

Self-check with standard data Calls:

SM4\_Encrypt; SM4\_Decrypt;

Called By: Input:

Output: Return:

1 fail ; 0 success Others:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int SM4\_SelfCheck()

{

int i;

//Standard data

unsigned char key[16] =

{0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,0xfe,0xdc,0xba,0x98,0x76,0x54,0x32,0x10};

unsigned char plain[16]=

{0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,0xfe,0xdc,0xba,0x98,0x76,0x54,0x32,0x10};

unsigned char

cipher[16]={0x68,0x1e,0xdf,0x34,0xd2,0x06,0x96,0x5e,0x86,0xb3,0xe9,0x4f,0x53,0x6e,0x42,0x46} ;

unsigned char En\_output[16]; unsigned char De\_output[16];

SM4\_Encrypt(key,plain,En\_output); SM4\_Decrypt(key,cipher,De\_output);

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

if ( (En\_output[i]!=cipher[i]) | (De\_output[i]!=plain[i]) ) {

// printf("Self-check error");

return 1; }

}

// printf("Self-check success"); return 0;

}