#include "main.h"

#include "aes.h"

#include "gcm.h"

#include "md.h"

#include "pkcs5.h"

#include "ecdh.h"

RNG\_HandleTypeDef hrng;

char input1;

char inputWords[70];

uint32\_t rngNumber;

uint32\_t tempNumber;

unsigned char secret\_key[256] = {0};

unsigned char hash\_key[128] = {0};

mbedtls\_aes\_context ctx;

mbedtls\_gcm\_context ctx\_gcm;

mbedtls\_md\_context\_t ctx1;

static const uint8\_t addition[4] = {0xfe, 0xa5,0x5a,0xef};

// Private function prototypes

static void MX\_RNG\_Init(void);

void getRandomKey(int length);

void getRandomKey128(int length);

void AESCTR(void);

void AESGCM(void);

int main(void)

{

    Sys\_Init();

    MX\_RNG\_Init();

    MX\_MBEDTLS\_Init();

    printf("\033[2J\033[;H");

    //printf("start\r\n");

    fflush(stdout);

    HAL\_Delay(1000);

    while (1)

    {

        printf("press 1 for AES-CTR, press 2 for AES-CTR\r\n");

        input1 = getchar();

        putchar(input1);

        printf("\r\n");

//      rngNumber = HAL\_RNG\_GetRandomNumber(&hrng);

//      printf("RNG: hex = %x, dec = %d \r\n",rngNumber,rngNumber);

        //getRandomKey(256);

        if (input1 == 49){

            AESCTR();

        }else if (input1 == 50){

            AESGCM();

        }

    }

}

void AESCTR(){

    mbedtls\_aes\_init(&ctx);

    char input[70] = {0};

    char output[70] = {0};

    char backoutput[70] = {0};

    unsigned char nounce\_counter[16] = {0};

    unsigned char stream\_block[16]   = {0};

    int lengths = 0;

    size\_t nc\_off = 0;

    while(1){

        // Generate random key

        getRandomKey(256);

        printf("Input a message: \r\n");

        lengths = uart\_getline(&USB\_UART,input,70);

        printf("Input message: %s\r\n",input);

        // Apply the key for encrypt

        if (mbedtls\_aes\_setkey\_enc(&ctx, secret\_key, 256) != 0){

            printf("Generate Key Failure!\r\n");

            return;

        }

        // Encrypt the plaintext

        if (mbedtls\_aes\_crypt\_ctr(&ctx, lengths, &nc\_off, nounce\_counter, stream\_block, &input, &output) != 0){

            printf("Encrypt the Plaintext Failure\r\n");

            return;

        }

        printf("The encrypted message is: %s\r\n", output);

        // clean the set

        memset(nounce\_counter, 0, sizeof(nounce\_counter));

        memset(stream\_block,   0, sizeof(stream\_block));

        nc\_off = 0;

        // Decrypt the message

        if (mbedtls\_aes\_crypt\_ctr(&ctx, lengths, &nc\_off, nounce\_counter, stream\_block, &output, &backoutput) != 0){

            printf("Decrypt the Plaintext Failure\r\n");

            return;

        }

        printf("The decrypted message is: %s\r\n", backoutput);

        // clear all the arrays

        memset(input,          0, sizeof(input));

        memset(output,         0, sizeof(output));

        memset(backoutput,     0, sizeof(backoutput));

        memset(nounce\_counter, 0, sizeof(nounce\_counter));

        memset(stream\_block,   0, sizeof(stream\_block));

        nc\_off = 0;

        printf("\r\n");

    }

}

void AESGCM(){

    // Initialization

    mbedtls\_gcm\_init(&ctx\_gcm);

    mbedtls\_md\_init(&ctx1);

    char input[20] = {0};   // input password

    char backoutput[70] = {0}; // Input message

    int lengths = 0;

    int messageLen = 0;

    char inputMessage[50];

    char messageEc[50];

    int correct = 0;

    // result of hashing

    unsigned char midkey[256] = {0};

    uint8\_t tagBuf[16];

    uint8\_t iv[12] = {1,2,3,4,5,6,7,8,9,0,1,2};

    // Generate random salt

    getRandomKey128(128);

    // get password

    printf("Input a password: \r\n");

    lengths = uart\_getline(&USB\_UART,input,20);

    printf("Password: %s\r\n",input);

    // Initialization of the hashing funciton

    extern const mbedtls\_md\_info\_t mbedtls\_sha256\_info;

    int resultCode = mbedtls\_md\_setup(&ctx1, &mbedtls\_sha256\_info, 1);

    // get the key for encrypt

    correct = mbedtls\_pkcs5\_pbkdf2\_hmac(&ctx1,input,lengths,hash\_key,128,5,256,midkey);

    if (correct != 0){

            printf("PBKDF2 derive failure\r\n");

            return;

        }

    // set the key for encrypt

    if(mbedtls\_gcm\_setkey(&ctx\_gcm, MBEDTLS\_CIPHER\_ID\_AES, midkey, 256) != 0){

        printf("Generate Key Failure!\r\n");

        return;

    }

    // clean the message buffer

    memset(inputMessage,           0, sizeof(inputMessage));

    // get the message

    printf("Input a message: \r\n");

    lengths = uart\_getline(&USB\_UART,inputMessage,50);

    messageLen = lengths;

    printf("Input message: %s\r\n",inputMessage);

    // Encrpyt the message

    if(mbedtls\_gcm\_crypt\_and\_tag(&ctx\_gcm, MBEDTLS\_GCM\_ENCRYPT, lengths, iv, 12, addition, 4,inputMessage, messageEc, 16, tagBuf) != 0){

        printf("Encrypt the Plaintext Failure!\r\n");

        return;

    }

    mbedtls\_gcm\_free(&ctx\_gcm);

    // clear the data buffer of key and input password

    memset(input,          0, sizeof(input));

    memset(midkey,         0, sizeof(midkey));

    // Get the password again

    printf("Input the password: \r\n");

    lengths = uart\_getline(&USB\_UART,input,50);

    printf("Password: %s\r\n",input);

    // Generate the key with same random salt

    correct = mbedtls\_pkcs5\_pbkdf2\_hmac(&ctx1,input,lengths,hash\_key,128,5,256,midkey);

    if (correct != 0){

        printf("PBKDF2 derive failure\r\n");

        return;

    }

    // Generate the key

    if(mbedtls\_gcm\_setkey(&ctx\_gcm, MBEDTLS\_CIPHER\_ID\_AES, midkey, 256) != 0){

        printf("Generate Key Failure!\r\n");

        return;

    }

    // Decrypt the message

    correct = mbedtls\_gcm\_auth\_decrypt(&ctx\_gcm,messageLen,iv,12,addition,4,tagBuf, 16,messageEc,backoutput);

    printf("Decrypt message: %s\r\n", backoutput);

    mbedtls\_gcm\_init(&ctx\_gcm);

    // clear data bufer

    memset(input,          0, sizeof(input));

    memset(midkey,         0, sizeof(midkey));

    memset(hash\_key,       0, sizeof(hash\_key));

    printf("\r\n");

}

// Generate random key

void getRandomKey(int length){

    for (int i =0; i<length; i++){

        if ((i+1)%32 == 1){

            rngNumber = HAL\_RNG\_GetRandomNumber(&hrng);

        }

        secret\_key[i] = rngNumber % 2;

        rngNumber = rngNumber / 2;

    }

}

// Generate random key

void getRandomKey128(int length){

    for (int i =0; i<length; i++){

        if ((i+1)%32 == 1){

            rngNumber = HAL\_RNG\_GetRandomNumber(&hrng);

        }

        hash\_key[i] = rngNumber % 2;

        rngNumber = rngNumber / 2;

    }

}

/\*\*

  \* @brief RNG Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_RNG\_Init(void)

{

    hrng.Instance = RNG;

    if (HAL\_RNG\_Init(&hrng) != HAL\_OK)

    {

        printf("Failed to initialize RNG\r\n");

        while (1);

    }

}