MAS433 Assignment

Wang Xueou (087199E16)

Exercise 1. Solution:

I. Finite Field Arthmetics

poly_mult.m: Performs the multiplication of two polynomials (**a** and **b**) in GF(2^8) using a third polynomial (**mod_pol**) for the modular reduction.

II. AES_128 Implementation

aes_demo.m:	**aes_demo** demonstrate the use of the AES_128 package. The call to **aes_init** supplies the actual en- and decryption function (**cipher** and **inv_cipher**) with expanded key schedule w, the substitution tables **s_box** and **inv_s_box,** and the polynomial matrices **poly_mat** and **inv_poly_mat**. These quantities have to be generated only once and can used by any subsequent en- or decipher.
aes_init.m	**aes_init** generates the two substitution tables **s_box** and **inv_s_box** by call to **s_box_gen** - defines the round constant vector **rcon** - defines a exemplary key and computes the expanded key schedule **w** - the two polynomial matrices used in **mix_columns** are also generated: **poly_mat** and **inv_poly_mat**
s_box_gen.m	This function creates substitution table **s_box** and **inv_s_box** used by the expanded key schedule and en- and decryption functions **cipher** and **inv_cipher** to directly substitute a byte by another byte of the same finite field (GF(2^8))
find_inverse.m	Find the inverse of b by brute force i.e., **b*b_inv=1** mod **mod_pol**. It loops through all possible byte values and stop once find the remainder is 1.
aff_trans.m	The affine transformation in the creation of **S-box**.
s_box_inversion.m	The inverse **S-box** is used in the decrypting function **inv_cipher** to revert the substitution in **S-box**.
rcon_gen.m	Round constant generating function used in Key expansion.

	It is a 4*4 matix of zeros except the 1st column.
key_expansion.m	Generate a 176 byte long key schedule **w**
rot_word.m	Perform the permutation to the word in the key expansion.
sub_bytes.m	Apply the **S-box** to one or more input bytes **bytes_in**
poly_mat_gen.m	The polynomial matrices **poly_mat** and **inv_poly_mat** are used in the **mixed_columns** function. Both matrices have a size of 4*4.
cycle.m	Perform the permutation in the functions shifting rows an **inv_shift_rows**. It cyclically permutes the rows of the input matrix. The first row is not shifted at all, the elements of the 2nd, 3rd and 4th row are shifted 1, 2 and 3 positions respectively to the direction specified.
cipher.m	Encrypt the plaintext
add_round_key.m	Perform a bitwise xor of the state matrix and the round key matrix
shift_rows.m	Shift rows according to the direction, where **state_in** is a 4*4 matrix
inv_shift_rows.m	Reverse the effect of the corresponding function **shift_rows** in the encryption process
mix_columns.m	Computes the new state matrix **state_out** by left-multiplying the current state matrix **state_in**. When encrypting, use the **poly_mat** generated in **poly_mat_gen** function When decrypting, use the **inv_poly_mat** generated in **poly_mat_gen**
inv_cipher.m	Decrypt the plaintext
disp_hex.m	DISP_HEX Display an array in hexadecimal form (it is a ready-to-use function)

When run the program, just run the main program file **aes_demo.m.**

Exercise 2. Solution:

1. One method is that, since the length of the IV is 128 bit. We can throw a fair coin 128 times, and record the results as a 0-1 sequence (0 for head, 1 for tail). In this way, we can get a perfectly random IV.

   In practice however, we use the deterministic random bit generator which is a FIPS-approved random number generator.

2. We can use the key to decrypt a short message. If the message is decrypted correctly, the key is right.