

# Exercise Sheet 1

## AES in Software

### Exercise 1 *Diffusion properties of AES*

*Diffusion* is mainly about making the ciphertext dependent on as many plaintext bits as possible. *Full diffusion* is attained when all ciphertext bits depend on all plaintext bits.

Consider AES encryption. How many rounds of the AES data transform (transformation of the data block) are needed to attain full diffusion in data?

### Exercise 2 *AES operations*

You have heard about AES-128 (referred to simply as AES in the sequel) and the basic underlying operations it uses in its 10 rounds: **SubBytes**, **ShiftRows**, **MixColumns** and **AddRoundKey**. In this exercise, we consider an AES implementation in software. You should not actually implement anything for this exercise.

1. Consider how to implement the AES in your favorite programming language.
  - a) How do you represent the AES state of 16 bytes?
  - b) How do you implement the **SubBytes** operation?
  - c) Recall the **ShiftRows** operation, where each byte of the  $i$ th row of the  $4 \times 4$  AES state is moved  $i$  positions to the left within the state, with  $i = 0, \dots, 3$ . How do you implement this?
  - d) For each column of the AES state, the **MixColumns** operation creates a dependency between the four input bytes and the four output bytes, by multiplying each column of the state by a  $4 \times 4$  matrix. How do you implement this?
  - e) Finally, think about the **AddRoundKey** operation. How do you implement this, and how do you store your round keys?
2. During encryption, the AES first uses the **AddRoundKey** operation to XOR a key to the state. This is followed by 9 rounds, and finally one special round, which does not involve the **MixColumns** operation. Not including the AES key schedule into the calculations, please determine:
  - (a) The number of byte lookups performed in one block encryption
  - (b) The number of byte multiplications performed in one block encryption

3. Estimate the complexity of the AES key schedule, in terms of (a), (b) and (c) above.

### Exercise 3 *C Implementation of AES functions*

In this exercise, you will implement the AES operations `SubBytes` and `MixColumns` in C. Make sure that each step works correctly before proceeding.

1. Implement `SubBytes` on the 16 byte AES state. The AES S-box lookup table is available on DTU Learn as `aessbox.c`.
2. Implement byte multiplication by 02 (in hexadecimal) in the AES finite field.
3. Implement `MixColumns` and its inverse on the 16 byte AES state. Use the multiplication by 02 to do this. The `MixColumns` matrix  $M$  and its inverse  $M^{-1}$  are given by

$$M = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \quad \text{and} \quad M^{-1} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}.$$

### Exercise 4 *MixColumns benchmarking*

Consider your implementation of `MixColumns` and its inverse from exercise 3. Measure their performances: Which one of them is faster? Why? You are welcome to use the benchmarking C file available on CampusNet as `timing.c`.

### Exercise 5 *C Implementation of T-tables*

Based on your solution to exercise 3, generate the T-tables  $T_0$ ,  $T_1$ ,  $T_2$ , and  $T_3$  for AES as explained in the lecture. How would you implement the last AES round which omits `MixColumns`?

### Exercise 6 *C Implementation of AES with T-tables*

You will now finish your AES implementation using the T-tables.

1. Implement the AES key-schedule and `AddRoundKey` on the 16 byte AES state.
2. Implement the full AES encryption/decryption. You can test your implementation using these test vectors (in hexadecimal):

Key	00000000000000000000000000000000
Plaintext	f34481ec3cc627bacd5dc3fb08f273e6
Ciphertext	0336763e966d92595a567cc9ce537f5e
<hr/>	
Key	10a58869d74be5a374cf867cfb473859
Plaintext	00000000000000000000000000000000
Ciphertext	6d251e6944b051e04eaa6fb4dbf78465