Exercise Sheet 1

AES IN SOFTWARE

Obligatory homework this week: None.
Optional homework this week: Exercise 3.

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. Study the diffusion properties of AES:

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

(b) Consider AES encryption. How many rounds of the AES key schedule are needed to attain full diffusion in the key?

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. 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
   (c) The number of XORs performed in one block encryption
2. 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 CampusNet 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

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

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 hexadecimals):

Key	`00000000000000000000000000000000`
Plaintext	`f34481ec3cc627bacd5dc3fb08f273e6`
Ciphertext	`0336763e966d92595a567cc9ce537f5e`
Key	`10a58869d74be5a374cf867cfb473859`
Plaintext	`00000000000000000000000000000000`
Ciphertext	`6d251e6944b051e04eaa6fb4dbf78465`