4 Key schedule

The key schedule function has been implemented in **AES.keySchedule**. The scheduled key computed by our method is

$$\begin{aligned} \text{Key} &= 011 \mid 100 \mid 100 \mid 001 \\ \text{Key round 0} &= 011 \mid 100 \mid 100 \mid 001 \\ \text{Key round 1} &= 001 \mid 101 \mid 001 \mid 000 \end{aligned} \tag{1}$$

5 Ciphering

We have implemented the different steps of the AES algorithm: ShiftRow, MixColumns and AddRoundKey.
Thanks to this functions (and to them inverse), we have implemented the cipher functions (and the decipher) in the java function class AES.cipher.
The result is:

$$\begin{aligned} \text{Original message} &= 001 \mid 001 \mid 110 \mid 011 \\ \text{Ciphered message} &= 010 \mid 001 \mid 110 \mid 110 \end{aligned} \tag{2}$$

6 Improve the efficiency of the AES

We can notice that each step of the AES algorithm is deterministic. We can also notice (by printing the successive keys and ciphered messages) that many combinations of key and messages are repeated.
Thus to make our algorithm more efficient, we have stored for each possible input message the result of the successive operations: subByte, shiftRow, and mixColumn.
To do so, we have needed for each input message (2^{12} different matrix) one only matrix of the same size. Thus the memory needed is 2^{12} times the size of a message. Which in our implementation represents a total memory of 65536 bytes.

We have on purpose ignored the addRoundKey operation because this operation depends on the key at each round. Hence, it would make the memory needed sky rocket.

The three operations listed bellow had a complexity of $O(n^2)$, where n is the size of the message. Thus by replacing each of them by a simple lookup in a table, the complexity of the algorithm would become constant (assuming that the xor operation needed for the addRoundKey is done in constant time).