Problem 1

Describe the role of cryptographic hash functions and MACs. How do they differ?

Problem 2

a) Given the recursive sequence/LFSR defined by

$$z_{i+4} = z_i + z_{i+1} + z_{i+2} + z_{i+3} \pmod 2$$

What are the periods using the keys

1 $K = 1000$?

2 $K = 0011$?

b) What are the periods with the same keys using the following LFSR?

$$z_{i+4} = z_i + z_{i+3} \pmod 2$$

Problem 3

Here you are to implement addition and multiplication in the *field* $GF(2^8)$. We can represent the elements as bytes. Addition is the same as bitwise xor.

To describe multiplication, we identify the byte $b_7b_6b_5b_4b_3b_2b_1b_0$ with the polynomial

$$b_7x^7 + b_6x^6 + \cdots + b_2x^2 + b_1x + b_0$$

with coefficients in $\mathbf{Z}_2$.

Mulitplication in $GF(2^8)$ is the same as polynomial multiplication, followed by reducing the product modulo the polynomial

$$x^8 + x^4 + x^3 + x + 1$$

This means that the 9 bit string 100000000, corresponding to x^8, is equivalent to (can be replaced by) the byte 00011011, corresponding to the polynomial $x^4 + x^3 + x + 1$. (Remember that the coefficients are mod 2, so -1 is the same as 1.) This byte is xor'ed with the remaining last 8 bits.

For example

$$\begin{aligned} 110011101 &= 100000000 \oplus 10011101 \\ &\equiv \ 00011011 \oplus 10011101 \\ &= \ 10000110 \end{aligned}$$

Multiplication by 00000010 is multiplication by x, which is just shifting all bits to the left, and appending 0 at end, followed by the reduction above if necessary. Multiplication by 00000100 (x^2) is multiplying by x twice, etc.

a) Implement the multiplication as code. Check your result against the supplied table.

b) Using lookup in the table in part a), find the multiplicative inverses of all the elements different from zero. Note that every element has a multiplicative inverse.

c) Check that the multiplication is commutative ($ab = ba$), associative ($(ab)c = a(bc)$ and distributive ($a(b + c) = ab + ac$, for all choices of a, b, c in $GF(2^8)$.

d) Can you find som element g in $GF(2^n)$ such that the sequence

$$g, g^2, g^3, ..., g^{255}$$

consists of all the non-zero elements in $GF(2^n)$?

Problem 4

We can construct a key-stream by using a block cipher in CTR-mode, by encrypting a sequence of values with a block cipher. The sequence consists of a nonce nonce n of 4 bits, which is concatenated with the counter, also 4 bits. The counter starts at 0, and add 1 to it for each round, up to $1111_2 = 15$

The encryption will take the byte $x = n||c$ and encrypt by taking the multiplicative inverse in $GF(2^8)$, and xor'ing it with the key.

Implement this in code.

a) With the key $k = 01001010$, and the nonce $= 0110$, write down the first 4 bytes produced.

b) What is the period of the key-stream? What properties of the encryption ensures that we do not get a shorter period?

c) Can the computation of the keystream be easily parallelized?

Problem 5

Vi define a HMAC as follows:

- Key $K = 1001$
- ipad $= 0011$
- opad $= 0101$
- h is the midsquare-hashing, calculationg $x^2 (\bmod 2^8)$ and retrieving the middle four binary digits. Eg. $1011^2 = 01111001$ (with leading 0), giving us 1110 as hash value.

a) Find the HMAC for the message 0110

b) You receive the message 0111, with HMAC 0100. Is it reason to believe that the message is authenic?

Problem 6

When defining cryptographic hash functions, one should in general have a *family* of hashes H_s, where the hashing function also depends on a random *salt* s. One way of achieving this is to prepend the salt to message, and then hash:

$$H_s(x) = H(s\|x)$$

When we use CBC-mode for encryption, we use a random initialization vector IV that is xor'ed with the first block of plaintext, before encryption.

a) What is the purpose of IV? What criteria are necessary for correct use of CBC-mode?

When using CBC-mode when creating a MAC, a CBC-MAC, we use a fixed IV, usually just 0. It may seem paradoxical, but it is important that one uses a fixed choice for IV. In particular, we cannot use the IV as salt.

Assume to the contrary, that one has not a fixed choice of IV, but uses it as a salt.

b) Show how an adversary can create valid CBC-MACs pairs from a known valid pair $(x, H_{IV}(x))$. Which messages x' can he create valid MACs for?

   NB! It does not depend on the hash function H. Recall also that the IV is known/in plaintext.