Homework 2

CSC 152/252 – Cryptography
Due: 11:59pm, Sunday, September 30

Read the document "Homework procedures" posted in Piazza resources for how to turn in your homework and policies on collaboration. A small sampling of your written homework and most of your programs will be graded over the semester. To minimize the chance that bad luck causes you a bad grade, do every problem to the best of your ability. Show your work. Ask questions if you need help.

What to turn in: Program files `hw2_mult.c` and `hw2_P52.c` and a single file with your written solutions named `hw2.pdf`. Misnamed files and files over 1MB will receive no credit.

Reading: AES chapter from *Understanding Cryptography* by C. Paar and J. Pelzl (`http://wiki.crypto.rub.de/Buch/en/download/Understanding-Cryptography-Chapter4.pdf`).

Written Problems:

1) Let's say that the key used with AES-128 is `0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F`. Compute the first two round keys used by AES-128 in this case (ie, compute k_0 and k_1 in Fig 4.2 which is also $W[0]$ through $W[7]$ in the Fig 4.5).

2) Using the k_0 and k_1 computed in Problem 1, what is the value of the evolving AES block after "round 1" in Fig 4.2 if initially the AES block ("plaintext x" in Fig 4.2) is `0xFF, 0xFE, 0xFD, 0xFC, 0xFB, 0xFA, 0xF9, 0xF8, 0xF7, 0xF6, 0xF5, 0xF4, 0xF3, 0xF2, 0xF1, 0xF0`.

3) GF(16) is defined like GF(256) except the polynomials all have degree less than 4 and the modulus is $x^4 + x + 1$. Calculate the following, each digit representing a field element in hexadecimal. (a) $5 + F$. (b) $5 - F$. (c) $5 \cdot F$. (d) $5/F$. Note that $5 - F$ is shorthand for $5 + (-F)$ where $-F$ is F's additive inverse, and $5/F$ is shorthand for $5 \cdot (F^{-1})$ where F^{-1} is F's multiplicative inverse.

Programming:

P1) I gave pseudocode in class for GF multiplication similar to the following.

```
Let p1 and p2 be elements of GF(2^n) (ie, polynomials of degree less than n)
extra = 0
Loop invariant: answer to original multiplication = (p1)(p2)+extra
while (p1 not 0)
    if (p1 has x^0 term)
        p1 = p1 - x^0
        extra = extra + p2
    p1 = p1 / x
    p2 = p2 * x
    if (p2 has x^n term)
        p2 = p2 - x^n + (modulus's lower terms)
Loop invariant: answer to original multiplication = (p1)(p2)+extra
```

The loop invariant is a statement that could be proven by induction to be true before and after each iteration of the loop. So, when the loop terminates because $p_1 = 0$, the loop invariant implies that the answer to the multiplication is in `extra`. Write a function in C that implements this pseudocode with the following header.

```
unsigned mult(unsigned a, unsigned b, unsigned modulus, unsigned degree)
```

The `modulus` parameter will represent a polynomial of degree `degree` which will be less than 32. Parameters `a` and `b` will each represent polynomials with degree less than `degree`.

Develop this pseudocode into a C function with the intended signature. Test it thoroughly by picking strategic inputs and comparing hand-calculated results with computer-generated results.

Put your code into a file `hw2_mult.c` and submit it via DBInbox. Your file should include only the required function and no `main`, should be appropriately documented, and should compile without warning or error when compiled using `gcc` or `clang` with compiler options `-std=c99 -W -Wall -Wpedantic -c`. The only headers your file is allowed to include are the standard ANSI C headers.

P2) I gave pseudocode in class for a 48-byte permutation. Since it's for our 152/252 class, I'll call it P52. Let 12 32-bit values be arranged as a 3-row by 4-column 2D array `s`.

```
P52(unsigned s[3][4])
    for i = 0 to 23
        for j = 0 to 3
            x = s[0,j] <<< 24
            y = s[1,j] <<< 9
            z = s[2,j]
            s[2,j] = x xor (z << 1) xor ((y and z) << 2)
            s[1,j] = y xor x xor ((x or z) << 1)
            s[0,j] = z xor y xor ((x and y) << 3)
        if (i mod 4 == 0)
            (s[0,0], s[0,1], s[0,2], s[0,3]) = (s[0,1], s[0,0], s[0,3], s[0,2])
            s[0,0] = s[0,0] xor 0x9e377900 xor (24-i)
        else if (i mod 4 == 2)
            (s[0,0], s[0,1], s[0,2], s[0,3]) = (s[0,2], s[0,3], s[0,0], s[0,1])
```

Write a function in C that implements this pseudocode with the following header.

```
void P52(unsigned s[12])
```

Note that `s` is not two dimensional. The author of the permutation thought it was clearer to explain its workings as if it was, but for efficiency it actually comes as a one dimensional array. You should consider the first four elements of the 1D array to conceptually be the first row of the 2D array, the next four elements the second row, and the last four elements the third row.

Put your code into a file `hw2_P52.c` and submit it via DBInbox. Your file should include the required function and no `main`, should be appropriately documented, and should compile without warning or error when compiled using `gcc` or `clang` with compiler options `-std=c99 -W -Wall -Wpedantic -c`. The only headers your file is allowed to include are the standard ANSI C headers.