Cryptographic Engineering: Assignment 2

Submit a C file through Canvas

For this lab, you are going to code some functions in C. A template (.c file) is created for you. Download it and complete the exercises below by filling the functions in the template. Compile the code and run it. At the end of the file, write **ALL** the output as a comment. Once completed, submit the C file through Canvas.

Remarks (These only apply to C code not Sagemath Code):

- Do **NOT** remove or add any function in the template.
- Do **NOT** modify the function name or parameters in the template.
- Do **NOT** remove, add, or modify any function in the template other than the ones stated below.
- Do **NOT** add any library to the C code.
- All variables you need have been created and initialized for you, you do NOT need any additional variable.
- Do **NOT** use any while loop or if statements. Only simple **for loops** are allowed `for(i=...;i<...;i++)` or `for(i=...;i>...;i-)`.
- Do **NOT** use / or % operators.

Use what you have learned in the Prelab to assist you in solving this assignment. A test bank (contains additional tests and answers) has been created for you. This test bank will make sure that you coded the functions correctly. The only thing you have to do is copy `assign#b_bank` file from Canvas into the same folder as where you run the compiled executable. It will show the values that failed. Once everything passes, it should display `bank_passed = 1`.

For exercises 1, 2, and 3, a test case is given. The inputs are the same and given below. They are 256-bit numbers in radix-2^{16} form with the least significant digit on the left.

```
a={0x13ab,0xfdb2,0xd231,0xc1b3,0xb1d0,0x0867,0x14d8,0x0102,
   0x5c99,0x380b,0x658a,0xc279,0x9b97,0x02d1,0xda40,0x52fa};
b={0x79f5,0x5500,0xda19,0xed75,0xe4a0,0x48b6,0x695f,0x0ee7,
   0x6bee,0x52ca,0xdfa5,0xaeac,0x9d49,0x03f1,0xeb9b,0xc4a4};
```

1. **Addition:** You are going to write a function that computes $r = a + b$ where a, b, and r are 256-bit numbers written in radix-2^{16}. Overflow bits should be eliminated.
   (a) In SageMath, compute $r = a + b$ and convert it into radix-2^{16}. Write the SageMath code where it says `SageMath Code 1 (a)`. Write the output in the variable `add_exp`.
   (b) Complete the function `add256`. You do not need any additional variables and the function should be completed in 1 for loop
2. **Subtraction:** You are going to write a function that computes $r = a - b$ where a, b, and r are 256-bit numbers written in radix-2^{16}. Overflow bits should be eliminated.
   (a) In SageMath, compute $r = a + b$ and convert it into radix-2^{16}. Write the SageMath code where it says `SageMath Code 2 (a)`. Write the output in the variable `sub_exp`.

(b) Complete the function `sub256`. You do not need any additional variables and the function should be completed in 1 for loop.

3. **Multiplication**: You are going to write three different functions that computes $r = a \times b$ where a and b are 256-bit numbers written in radix-2^{16} and r is 512-bit numbers written in radix-2^{16}. Similar to the prelab, you should not propagate the carry at any step. The output you should get from each function (part a, b and c) is written for you in the variable `mul_exp`.

   (a) Complete the function `schoolbook_mul256`. You do not need any additional variables and the function should be completed in 1 nested for loop.

   (b) Complete the function `comba_mul256`. You do not need any additional variables and the function should be completed in 2 nested for loops.

   (c) Complete the function `karatsuba_mul256`. You do not need any additional variables. Compute the variables in the same section together in the same for loops. Use the least number of loops. Reminder to not propagate any value including the addition and subtraction.

4. In this exercise, you are going to get the number of cycles for each method used in Exercise 3. The method that has less number of cycles is faster. The function to compute the number of cycles has already been created for you.

   Remark 1: The only requirement is to log the number of cycles.

   Remark 2: Make sure to run the code multiple times as sometimes your PC is running other tasks and the cycle count will not be accurate.

Grading

Exercise	Grade	Note:
Prelab	20 pts	Full grade for serious attempt even if answer is incorrect.
1 (a)	4 pts	
1 (b)	10 pts	
2 (a)	4 pts	
2 (b)	10 pts	
3 (a)	15 pts	
3 (b)	15 pts	
3 (c)	20 pts	
4	2 pts	To receive the full points, the actual faster function must have less clock cycles
Formatting		-5 pts. To not lose any points, readability and proper indentation are required. Follow the same formatting as the template. Indent 4 spaces when going into the body of a block. Leave an empty line when working on a new block or when you are working on a new part in your function.
Late Submission		-5 pts per day for 2 days. Afterwards a grade of 0 is given.
Total	100 pts	

Remark 1: Code that does **NOT** compile on Linux (gcc) will receive a grade of 0 for the whole assignment.

Remark 2: Students are not allowed to modify any function other than the ones stated in the exercises. Modifying the function name or parameters will receive a grade of 0 for that function. Modifying functions not stated in the exercises will recieve a grade of 0 for the whole assignment. Adding any library (such as math.h) will receive a grade of 0 for the whole assignment.

Remark 3: A few points might be deducted if the code is ambiguous. Provide comments whenever the code is not clear or too complex. A good rule of thumb to know when to comment is to ask these 2 questions: Will you be able to quickly understand and update this part of your code if you looked at it in 5 years or if someone else wrote the code, will you be able to understand it in 5 years.