Cryptographic Engineering: Assignment 3

Submit a pdf file and a c file through Canvas.

Assignment A

1. **Montgomery Multiplication:** Write a SageMath code that computes (`0x486d1` $\times$ `0xd1dc3` + `0x138a4` $\times$ `0x47f45` − `0xbdcc8`) mod p. $p = 2^{20} - 3$. Below is a SageMath code that will help you compute the result. Copy it to the top of your SageMath code and use the functions to compute the result. Do NOT use any other function or arithmetic operation ($+$, $-$, $*$, >>,&...). In addition p, R, $R2 = R^2 \mod p$, and $Rinv = R^{-1} \mod p$ have been defined for you. Print the output in hexadecimal. You receive the full grade, you should submit the code and the final output as a comment in the last line.

```
p=2^20-3
R=2^20
R2=(R^2) % p
Rinv=inverse_mod(R,p)
def mod_add(a,b): return (a+b)%p
def mod_sub(a,b): return (a-b)%p
def mont_mult(a,b): return (a*b*Rinv)%p
```

Function	Implements	Note
`mod_add(a,b)`	$(a + b) \mod (p)$	
`mod_sub(a,b)`	$(a - b) \mod (p)$	
`mont_mult(a,b)`	$(a \times b)R^{-1} \mod (p)$	$R = 2^{20}$

2. **Barrett reduction:** Following the Barret reduction algorithm, compute μ and **write the output for each line** of the algorithm $(q_1, q_2, q_3, r_1, r_2, r)$ in decimal for $x = 10256$, $p = 71$, and $k = 7$. You can compute it by hand but if you use SageMath, flooring can be done using // operator instead of / operator. Note that r is the output in 3 different lines. Show them all.

Assignment B

For this assignment, 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.

A test bank (contains 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 to the same folder as where you run the compiled executable. It will show whether each question passed or failed. You can disable testing for a specific exercise by switch #define EXn 1 to #define EXn 0. Make sure to re-enable all of them once completed.

Similar to the previous assignment, we will work with radix-2^{16}. To not spend a lot of time working on re-implementing multi-precision arithmetic, some functions have been provided for you in the table below. All functions will make the final result in radix-2^{16} except the most significant digit. All overflow bits will be stored in the most significant digit. Please do not modify the variables `i` and `carry` or the functions will not work properly.

Function	Implements
`ADD_LOOP(a,b,r)`	Computes $r = a + b$
`SUB_LOOP(a,b,r)`	Computes $r = a - b$
`ADD_MASK_LOOP(a,mask,b,r)`	Computes $r = a +$ (`mask`&`b`)
`CARRY_PROP_LOOP(a,r)`	Carry propagates a into r, aka converts all array elements to radix-2^{16}
`ADD_MUL_DIGIT_LOOP(a,b,c0,r)`	Computes $r = a + b \times c0$ where $c0$ is 1 digit.

If you want to add two numbers but let's say you want the number start from index 16 as first digit (`a[16],a[17],...,a[31]`), then you can pass `a+16` as an argument.

We are going to work with modulus $p = 2^{255} - 19$. The prime is provided to you in the c code in the variable PRIME. This is a pseudo-Mersenne prime currently used in many cryptography algorithms.

The schoolbook multiplier from the previous assignment is now in function `mul256`. Combing it with the pseudo-Mersenne reduction, which you will implement, will allow us to implement modular multiplication (`mod_mul`) and modular squaring (`mod_sqr`). These two functions are provided to you and will help you implement modular inversion. Your final result for all questions has to be in radix-2^{16}.

1. Implement a secure and efficient $(a + b) \mod p$ where $0 \leq a < p$ and $0 \leq b < p$. To be filled in function `mod_add`.
2. Implement a secure and efficient $(a - b) \mod p$ where $0 \leq a < p$ and $0 \leq b < p$. To be filled in function `mod_sub`.
3. Implement a secure and efficient $a \mod p$ using pseudo-Mersenne technique where $0 \leq a < p^2$. To be filled in function `psu_reduce`.
4. Implement a secure and efficient $a^{-1} \mod p$ where $0 \leq a < p$. To be filled in `mod_inverse`. Hint: In binary, $p - 2$ is 250 1s, then 0, then 1, then 0, then 1, then 1.

Grading

	Exercise	Grade	Note:
Assignment A	1	15 pts	
	2	15 pts	
Assignment B	1	15 pts	
	2	15 pts	
	3	20 pts	
	4	20 pts	
	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		No late submission allowed. Grade of 0 will be 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.