COMP8760

A1: Programming Assignment

Sanjay Bhattacherjee
(s.bhattacherjee@kent.ac.uk)

Submission deadline: 23:55 hrs on Tuesday, 12^{th} November, 2024

Extension: If you face extenuating circumstances and would like to request for an extension, please apply via KentVision as with any other module. In case you get an extension, your deadline for submission will be 23:55 hrs on Tuesday, 19^{1h} November, 2024.

What do you submit? You will have to submit **one zip file** following the specified convention below.

- Your zip file should be named as: <your ID>.zip. For example, if your ID is ab123@kent.ac.uk, the name of your file should be "ab123.zip".
- Your zip file should contain six files - each corresponding to one task. These files should each be named after the respective task. For example, if you have written the programs in python and taken screenshots as jpeg images, then your files should be named as:

  "ab123_task1.py",
  "ab123_task2.py",
  "ab123_task3.py",
  "ab123_task4.py",
  "ab123_task5.jpg" and
  "ab123_task6.py".

Where to submit? Using the link on Moodle.

General Instructions:

- You have to complete six tasks as described in this sheet. Tasks 1, 2, 3, 4 and 6 are programming tasks. Each program should be written in a single separate file as described above, so that it can be directly executed using command-line arguments. You may use C/C++, Java or Python (>=3.0) to do these tasks.
- Each program should provide an input-output interface in the precise format as detailed under **Sample I/O** respectively. If you do not follow the format, marks may be deducted. The blanks denoted as _____ in the **Sample I/O** will be substituted with actual values while running the program.
- You should not use pre-written programming libraries other than those for standard input-output or generating (pseudo-)random numbers. For exponentiation, you should not use any in-built operators either, but your own algorithmic implementation of the operation instead.

Marking criteria: The total marks for this assessment has been distributed among the six tasks. The mark allotment for each task has been indicated at the beginning of the task description. You will be marked based on the following criteria.

- Syntactical correctness: If your code encounters compilation / interpreter errors you will be marked with zero for that task.
- Functional correctness: If your code works correctly for all input values it is tested with, you will qualify to get 0.8 fraction of the mark allotted to that task. Correctness will be judged based on your adherence with the **Sample I/O** format specified above as well as the correctness of the computations done by the program. You will receive partial marks if you have not met all the requirements of the task.
- Code quality: To qualify for full marks in a task, your code has to be well commented and its functionality explained.

Task 1: Exponentiation. Marks: 10%
The program will take as input positive integers m, e and N. It will provide as output the value of $m^e \mod N$ using not more than $\log e$ multiplications.
Sample I/O:

```
Please enter m:  _____
Please enter e:  _____
Please enter N:  _____
----------------------
The value of m ∧ e mod N is:  _____
----------------------
```

Task 2: Extended Euclidean algorithm (EEA). Marks: 20%
The program will take as input two integers a and b and provide as output three numbers g, x and y such that

$$\gcd(a, b) = g = xa + yb.$$

Sample I/O:

```
Please enter a:  _____
Please enter b:  _____
----------------------
The values of (r, s, t) in the steps of EEA are:
(_____, _____, _____)
(_____, _____, _____)
⋮
(_____, _____, _____)
----------------------
The values of (g, x, y) are:  (_____, _____, _____)
----------------------
```

Task 3: Finding a prime number. Marks: 20%
The program will take as input a parameter n and provide as output a n-bit prime. The output number p should be in the range $2^{n-1} \leq p < 2^n$
Sample I/O:

```
Please enter the parameter n:  _____
----------------------
The n-bit prime is p = _____
----------------------
```

Task 4: "Naive" RSA encryption system implementation. Marks: 20%
The program will take as input the security parameter ν. (Your program should work with $\nu \approx 20$. In other words, your program will not be tested for non-standard integer sizes ($\geq$ 32 bits). That said, if you feel adventurous, please visit the GNU Multiple Precision Arithmetic Library for more details.)

Your program generate the two $\nu/2$-bit primes, and the integers N, e and d. It will then prompt the user to choose one of the two menu options - encryption and decryption. If the user chooses encryption, the program will prompt the user to enter an element from the plaintext space $\mathbb{Z}_N$ and provide its encryption. If the user chooses decryption, the program will prompt the user to enter an element from the ciphertext space $\mathbb{Z}_N$ and provide its decryption.
Sample I/O:

```
Please enter the security parameter 'nu':  _____
----------------------
Setup:
The first prime generated by the Setup algorithm is p = _____
The second prime generated by the Setup algorithm is q = _____
The integer N = pq = _____
The encryption exponent is e = _____
The decryption exponent is d = _____
----------------------
```

```
Please enter an option:
1 to Encrypt
2 to Decrypt
Any other number to quit
Your option:  _____
----------------------
Encryption:
Your message space is the set {Z/NZ} = {0, 1, ..., _____}
Please enter a number from this set:  _____
The ciphertext for your message _____ is _____
----------------------
Please enter an option:
1 to Encrypt
2 to Decrypt
Any other number to quit
Your option:  _____
----------------------
Decryption:
Your ciphertext space is the set {Z/NZ} = {0, 1, ..., _____}
Please enter a number from this set:  _____
The plaintext for your ciphertext _____ is _____
----------------------
```

Task 5: IND-CPA security. Marks: 10%
You will have to submit **one image file for this task**. The image file should contain the screenshot of appropriate executions of the programs you have written for Task 4.

We recollect that if an encryption scheme encrypts a plaintext message to the same ciphertext every time, it is insecure against an IND-CPA adversary. Using the program you have written for Task 4, provide an example input-output demonstrating the insecurity of "Naive" RSA with respect to an IND-CPA adversary.

Task 6: IND-CCA security. Marks: 20%
Write a program to demonstrate that "Naive RSA" is insecure with respect to an IND-CCA adversary. The program will take as input a public key (N, e) and a ciphertext c created by the program written for Task 4. It will then output the modified ciphertext $c' = 2^e \cdot c \pmod{N}$. It will then output the element $2^{-1} \pmod{N}$. It will take as input the message m' found by decrypting c' using the program written for Task 4. It will finally output the original message m that was encrypted to c.
Sample I/O:

```
Please enter the public parameter N:  _____
Please enter the encryption exponent e:  _____
----------------------
Please enter the ciphertext c:  _____
----------------------
The modified ciphertext c' is = _____
The inverse of 2 mod _____ is = _____
----------------------
Please decrypt the modified ciphertext c' using your program from Task 4.
Please input the plaintext m' decrypted from c':  _____
The original plaintext message m computed from m' is:  _____
----------------------
```