Challenge Sheet 3 (10 Points)

Date due: December 12th 2022

Submission Instructions

Solutions to this challenge sheet must be submitted by **December 12th 2022** *before 11:59 am* to the CMS. **Make sure that your solution contains your name and matriculation number at the top of the files!** It is not allowed to submit these solutions in a group, as you do with the regular exercise sheet solutions. In the .py files, you may add as many additional methods as you want, but please do not rename the existing methods or add code outside of methods. How many points you get for the tasks is determined by tests on our system. You can also leave some feedback about this sheet here: Survey 3

1 Rivest-Shamir-Adleman

This task deals with the very famous RSA algorithm.

(1 point) (a) We start simple with the implementation of **encrypt(...)** and **decrypt(...)** in **rsa.py**. Both only have to deal with integers as plaintext and ciphertext and not with bytes or strings as before. Also you may assume that your program will only face small e's, so you don't need to implement something like shown on the slides *Exponentiation in Modular Arithmetic*. Nevertheless, you are welcome to try and implement handling large e's if you want to play around a bit.

(3 points) (b) Now that the encryption and decryption are complete, we can prepare the calculation of the keys. For RSA, we need to know the greatest common divisor of two numbers. So let's begin by implementing **gcd(...)** in **rsa.py**.

(4 points) (c) Next, we also need the **extendedEuclideanAlgorithm(...)** in **rsa.py**. This task is probably a bit more difficult and requires some experimentation. There are many creative ways to implement this, so we'll leave it up to you how you want to do it. The only important part is, that the method should return a tuple with the three items d, a and b, where $d = x \cdot a + y \cdot b$.

Note that d is not the same d used in RSA. It refers to divisors.

(2 points) (d) **This task can only be solved if you have already completed b) and c)!**
Now you have everything you need to implement **calculateKeys(...)** in **rsa.py**, which should return pk (public key) and sk (secret key). To do so, you need to calculate both N as well as the *Euler's totient function*. After that you can use *extendedEuclideanAlgorithm(...)* from c) to find d. Moreover, you can assume that p and q are primes, but you have to check whether e is valid or not. For that purpose you can use *gcd(...)* from b). If e is not valid, please return *None* instead of pk and sk. In case you want to play around a bit more, you can also

try to check if p and q are prime numbers. But again, this is not necessary for this task!

Hint: Test your program with values other than those we provide. It might help to find a common mistake related to d of sk!