Prof. Dr. Heiko Knospe **Summer Semester 2020**

Cryptography Lab 1

Preparation: Recapitulate the chapters *Fundamentals of Cryptography* and *Elementary Number Theory* and get familiar with the basics of Python programming.

The Lab uses the mathematics software SageMath and Jupyter notebooks.
(`http://www.sagemath.org`).

Make sure to evaluate your input (Shift-Enter) and to start a new cell **after each command** or after each definition of a function. **Do not write** multiple commands in a single cell unless it is really necessary!

Using the code or parts of it or adapting the code of other students is **not acceptable** and is considered cheating with consequences.

Replace X by your student matriculation number (8 digits starting with a 1).

1. Start SageMath, open a new worksheet and rename it to X. Write your full name and X into the first cell:

```
# Name: ...
# Matriculation Number: ...
```

Perform some elementary modular operations using Sage. Do not forget to use new cells (Shift-Enter) and to save your worksheet regularly. Set $n = XXX$ (three times the matriculation number), $a = X19898X65432098772$ and $b = 6543209877123456789143333 6543209877$.

(a) Find the prime factor decomposition of n, a and b. Use the `factor` command.

(b) Compute $a + b \bmod n$, $ab \bmod n$, $a^b \bmod n$, using `mod(..,..)` and `power_mod(..,..,..)`. Try to compute $a\hat{}b$ and `mod`$(a\hat{}b, n)$. Why does this problem not occur with `power_mod(..,..,..)` ?

(c) Are a or b invertible modulo n ? Why or why not ? Compute a^{-1} or $b^{-1} \bmod n$, using `mod(1/..,..)`.

2. Implement the Extended Euclidean Algorithm (see the chapter on elementary number theory). Complete the definition of the Sage function `exteucl` listed below. Note that correct indentation is extremely important in Python.

```
def exteucl(a,b):
    x0=1;x1=0;y0=0;y1=1;sign=1;
    while b!=0:
        r=a%b
        q=a//b
        ...
    ...
    return(gcd,x,y)
```

Test the function with `exteucl(845,117)`. The Euclidean Algorithm computes the remainders of successive integer divisions. Extend the code so that the sequence of remainders is printed out. Compute `exteucl(a,n)` and `exteucl(b,n)` using the numbers in Task 1. Set `[g,x,y]=exteucl(a,n)` and verify that $g = ax+ny$ and $g \equiv ax \bmod n$. Comment out the print statement after completing this task.

3. a) Write a function `numberofunits` which counts the number of the invertible integers mod n and prints out the result.

```
def numberofunits(n):
     for i in range(1,n):
          if exteucl(i,n)[0]==1:
            ...
```

Give the number of units mod n where $n = X - 10000000$. Compare the result with the built-in function `euler_phi`.

b) Write a function `someunits` which prints the first 100 units mod n. If there are less than 100 units modulo n, print all units. Test your function with $n = 97$, $n = 400$ and $n = X$.

4. Set up an RSA cryptosystem and encrypt a message. First, generate two 1024-bit random primes. This may take some time.
`p=random_prime(2^1024)`
`q=random_prime(2^1024)`
Set $n = pq$ and $phi = \varphi(n) = (p-1)\cdot(q-1)$. Print out p, q, n and $\varphi(n)$.
Print out all integers $1 < e < 100$ with $\gcd(e, \varphi(n)) = 1$. Choose $e > 1$ as small as possible such that $\gcd(e, \varphi(n)) = 1$. Then (e, n) is your public RSA key. Compute your private RSA key:

$$d \equiv (e \bmod \varphi(n))^{-1}$$

Verify that $ed \equiv 1 \bmod \varphi(n)$. Print out d. Now suppose that $m = XXXXX$ is the given plaintext, i.e., five times your matriculation number. Compute the RSA ciphertext $c = m^e \bmod n$. Decrypt the ciphertext by computing $c^d \bmod n$ and compare the result with the plaintext m.

Hint: SageMath distinguishes between residue classes and integers. Use `ZZ(d)` to transform the residue class d into an integer.

Remark: The above schoolbook RSA algorithm has weaknesses. Randomized versions of RSA should be used instead.

5. Write a SageMath function which generates $\lfloor \frac{X}{100} \rfloor$ random integers less than 2^{1024}, checks their primality and counts the number of primes. Then determine the expected number of primes using the Prime Number Theorem and compare this with your experimental result.

Hints: A random integer less than 2^{1024} can be generated with `ZZ.random_element(2^1024)`. The primality can be tested using the fast function `is_pseudoprime`. SageMath usually returns symbolic values. The numeric value of a real number can be obtained with `RR(...)`, e.g., `RR(log(2))` gives $\ln(2)$.

Completion: Fully document your code by using the # character. Make sure that all cells are evaluated; then save the notebook. Upload the file `X.ipynp` to Moodle.