Network Security

Assignment: Cryptography

One-time pad [50 points]

Warm-up

Let set A be $\mathbb{Z}/n\mathbb{Z}$, i.e., a finite cyclic group of integers modulo n. Assume that A represents the alphabet used for plaintext generation:

$$A = \{0, 1, 2, ..., n-1\}$$

There is a binary operation of addition defined on A, namely: $x \mapsto x+k \pmod{n}$, where $x, k \in A$. Assume a brute-force attack on a text generated from A and encrypted with a one-time pad using the operation of addition. A one-time pad is also generated from the same alphabet. Implement a function that takes as parameters n and $lngth$, where $n = |A|$ and $lngth$ is the length of the ciphertext string. The function should return the number of all possible strings of length equal to $lngth$ generated from A.

The signature of the function should be the following:

```
def computeNumberOfStrings(n, lngth)
```

Compute the number of all possible strings for the following $lngths = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ and $|A| = 26$ and store the resulting values in a separate array y.

Recall that any exponential with the base b can be expressed in terms of the base e:

$$cb^x = ce^{x \ln b} \tag{1}$$

Fit the corresponding curve with $x := lngths$ and y in the following way:

```
import numpy as np
from scipy.optimize import curve_fit

def fitCurve(x, y):
    def func(x, c, d):
        return c * np.exp(d * x)

    params, _ = curve_fit(func, x, y, p0=(1., 1.))

    return params.tolist()
```

The function above should return the fitted params, i.e., **c** and **d**.

Plot and observe the resulting graph with the precomputed data and fitted curve.

Compute the exact values of curve parameters **c** and **d** and return them from the following function:

```
def exactParams(n):
    # compute/return the exact params c and d
```

```
    return [c, d]
```

Goal

Your goal is to produce a Python3 script called `warm_up.py`, which accepts three arguments separated by a space: the first argument is always a name of the called function (e.g., `fitCurve`), the rest of the parameters are dependent on the function to be called, i.e., for `fitCurve` the additional parameters would be `x` and `y` provided in quotes as comma-separated strings, for `computeNumberOfStrings` - `n` and `lngth`, and one parameter `n` for `exactParams`. The script should produce either a single value in case of `computeNumberOfStrings` or a comma-separated string in case of both `fitCurve` and `exactParams`. The script should produce "ERROR" in case of exception/error/failure or any noncompliance with the described use cases.

```
python3 warm_up.py "computeNumberOfStrings" 26 0
1
python3 warm_up.py "computeNumberOfStrings"
ERROR
```

Submission Instructions

After completing the assignment, you should only submit the resulting script. A `requirements.txt` file should be provided if you use any modules from the Python Package Index (pip).

OTP implementation

Let A be the alphabet and $lngth$ be the length of the string with the plaintext s.t. $A = \{0, 1, 2, ..., 255\}$ (i.e., A represents ordinal values of ASCII symbols) and $lngth \leq 50$.

Implement one-time pad encryption with the following randomly generated key of capital English characters:

```
one_time_chars = ['B', 'H', 'Z', 'A', 'P', 'S', 'I', 'E', 'Z', 'S', 'L', 'A', 'G',
'V', 'C', 'E', 'X', 'L', 'E', 'U', 'F', 'X', 'X', 'X', 'G', 'O', 'F', 'J', 'L', 'D',
'H', 'S', 'O', 'S', 'C', 'O', 'Q', 'O', 'J', 'G', 'X', 'W', 'W', 'P', 'R', 'Z', 'X',
'D', 'M', 'M']
```

Note that the key should be converted to the ordinal values before encryption to be compliant with the alphabet.

Apply a bitwise involutory modulo 2 addition for encryption per each byte of the key k and plaintext t to get a ciphertext c:

$$c_i = t_i \oplus k_i \quad \forall \quad 0 \leq i < lngth$$

Goal

Your goal is to produce a Python3 script called `otp_capital.py`, which accepts one argument: a string of ASCII alphanumeric characters to be encrypted. The script should produce the hexadecimal ciphertext string as an output.

```
python3 otp_capital.py "Encrypt me"
0726393329233d653736
```

The script should produce "ERROR" in case of exception/error/failure or any non-compliance with the described use cases.

Submission Instructions

After completing the assignment, you should only submit the resulting script. A `requirements.txt` file should be provided if you use any modules from the Python Package Index (pip).

Reused OTPs

One-time pad can be cracked if the key is used more than once to generate several different ciphertexts. Suppose that one intercepts two messages enctypted with the same key k:

$$c^{(1)} = t^{(1)} \oplus k$$
$$c^{(2)} = t^{(2)} \oplus k$$

Then one can compute:

$$c^{(1)} \oplus c^{(2)} = (t^{(1)} \oplus k) \oplus (t^{(2)} \oplus k) = t^{(1)} \oplus t^{(2)}$$

Recall that it holds for any ASCII English character that:

```
>chr(ord('A') ^ ord(' ')) == 'a'
>True
>chr(ord('B') ^ ord(' ')) == 'b'
>True
>chr(ord('C') ^ ord(' ')) == 'c'
>True
...
>chr(ord('a') ^ ord(' ')) == 'A'
>True
>chr(ord('b') ^ ord(' ')) == 'B'
>True
>chr(ord('c') ^ ord(' ')) == 'C'
>True
...
```

Now assume that you have managed to intercept several messages encrypted with the same key. Implement an algorithm that deduce the key based on the ciphertexts available in the corresponding file `otp_ciphertexts.txt`. The final goal is to decrypt the following message:

```
222c326d2535213b662038263532643b3a2e78252161273c2b2738362136
```

Goal

Your goal is to produce a Python3 script called `otp_reused.py`, which should produce the decrypted message.

```
python3 otp_reused.py
HERE GOES YOUR DECRYPTED MESSAGE
```

Note that the results should be precomputed prior to running the script so that no calculations are performed while the script is called. The script should produce "ERROR" in case of an exception/error/failure or any non-compliance with the described use cases.

Submission Instructions

After completing the assignment, you should only submit the resulting script. A `requirements.txt` file should be provided if you use any modules from the Python Package Index (pip).

Discrete Logarithm Problem [50 points]

Let G be a finite Abelian group. The discrete logarithm problem (or DLP) is posed to find an integer x such that $g^x = h$ for known $g, h \in G$, if such an integer exists.

Continuous Logarithm Problem

Let us first consider a simpler case: the *continuous* logarithm problem, i.e., the case when $g, h, x \in \mathbb{R}$ with the standard operation of exponentiation on reals. In such a case, the value of x can be easily calculated. Implement the function that takes two parameters h and g, and returns x.

```
def solveContinuousLog(h, g)
```

Goal

Your goal is to produce a Python3 script called `cont_log.py`, which accepts two arguments: h and g. The script should produce the value of x as an output.

```
python3 cont_log.py 8 2
3
```

The script should produce "ERROR" in case of exception/error/failure or any non-compliance with the described use cases.

Submission Instructions

After completing the assignment, you should only submit the resulting script. A `requirements.txt` file should be provided if you use any modules from the Python Package Index (pip).

DLP under addition

Let G be a finite Abelian group of integers modulo n with a binary operation of addition. Note that in such case, the problem of discrete log boils down to finding x with known $g, h \in \mathbb{Z}/n\mathbb{Z}$ in the following equation: $h = x \cdot g$

Goal

Your goal is to produce a Python3 script called `solve_dlp_add.py`, which accepts three arguments: h, g, and n. The script should produce the value of x as an output.

```
python3 solve_dlp_add.py 10 2 11
5
```

The script should produce "ERROR" in case of exception/error/failure or any non-compliance with the described use cases.

Submission Instructions

After completing the assignment, you should only submit the resulting script. A `requirements.txt` file should be provided if you use any modules from the Python Package Index (pip).

ELGamal encryption

Implement ElGamal encryption using a function that takes four parameters g for generator, h for public key computed by your communicating counterpart, n for the modulo operations, r for an ephemeral random key, and t for the plaintext to be encrypted. The function should return two values: c_1, c_2, where $c_1 = g^r$ and $c_2 = t \cdot h^r \mod n$.

```
def elGamalEncrypt(h, g, n, r, t)
# encryption implementation
return c1, c2
```

In addition implement ElGamal decryption using a function that takes n for modulo operations, a for a private key used for the generation of h, and finally - values computed by encryption function: c_1 and c_2. The function return the decrypted message.

```
def elGamalDecrypt(n, a, c1, c2)
# decryption implementation
return message
```

Goal

Your goal is to produce a Python3 script called `el_gamal.py`, which accepts five arguments: the first argument is always a name of the called function (e.g., `elGamalEncrypt`), the rest of the parameters are dependent on the function to be called. The script should produce the comma-separated string in case of encryption and the decrypted message in case of decryption.

```
python3 el_gamal.py "elGamalEncrypt" 65 3 809 89 "d"
345, 517
```

The script should produce "ERROR" in case of exception/error/failure or any non-compliance with the described use cases.

Submission Instructions

After completing the assignment, you should only submit the resulting script. A `requirements.txt` file should be provided if you use any modules from the Python Package Index (pip).

Make sure your assignment conforms to the in- and output described in the Goal subsection. Your assignment is partly graded through an automated system. Any deviations from the described in- and output can affect your grade.