Exercises for Module 1

The exercises for the first module are about Python basics, exclusive OR, and random number generation.

Python

Exercise 1: Write a Python program which takes a string as input, and prints the individual byte values of the string

Exercise 2: Write a Python program which takes a positive integer as input and prints the binary and hexadecimal representation of this integer.

Exercise 3: Write a Python program which takes a text string as input and saves it as UTF-8 encoded variable. Test if it works when using Danish letters such as æ, ø, å.

Exercise 4: Repeat the previous exercise but encode the string in ASCII instead of UTF-8. Does it work? Why or why not?

Exercise 5: Write a Python program that opens a PNG file in binary mode and prints the first 8 bytes of the file (its signature). Compare with https://en.wikipedia.org/wiki/Portable_Network_Graphics#File_header

Cryptography (basics)

Exercise 6: Let plaintext = 0b1100 and key = 0b1001. (Note: in Python, you can define an integer by using e.g. x = 0b1111 (which will create a variable called x with the value 15). Also note that XOR is only defined between two integers.

(a): Compute the XOR of plaintext and key, call this the ciphertext.

(b): Compute the XOR of ciphertext and the key. What do you observe?

(c): Change one bit of the ciphertext, and compute the XOR of it and the key. What happens to the plaintext? (Note: This property is called malleability.)

Exercise 7: On paper and in Python, compute the following remainders (note that the remainder must always be positive):

(a): 5 mod 21

(b): 17 mod 11

(c): -27 mod 23

Exercise 8: Why is the OTP not used for communication over the Internet?

Exercise 9: Implement a fixed-XOR function using the instructions here: https://cryptopals.com/sets/1/challenges/2

Random number generation

Exercise 10: Consider the following Python code for simple random number generation:

```
import random, time

current = time.time()

random.seed(current)

r1 = random.randrange(0, 65534)

random.seed(current)
```

r2 = random.randrange(0, 65534)

print(r1)

print(r2)

Does this code produce consecutive random numbers? If not, suggest an improvement.

Exercise 11: Study which random number generators are available in the standard library of a programming language of your choice? Are cryptographically secure options available?

Exercise 12: What are the advantages and disadvantages of using an RNG instead of an PRNG?

Exercise 13: Implement a naive random number generator for generating numbers in the range 0 to 191, as follows: Start by generating a random 8-bit value and interpreting it as an integer. Then reduce this integer modulo 192. Generate many integers in this way and check the probability distribution of the resulting numbers.

Exercise 14: Read the webpage https://inversegravity.net/2019/password-entropy/ about entropy of passwords. What is the main takeaway of this article?

Note: Exercise 10 was adapted from https://cryptobook.nakov.com/secure-random-generators, and Exercises 11 – 14 were adapted from Cryptography Engineering by Ferguson, Schneier and Kohno.

Cryptography (advanced)

The remaining exercises are more difficult; solve them after you've solved the above ones.

Exercise 15: Implement a single-byte XOR cipher using the instructions here: https://cryptopals.com/sets/1/challenges/3

Exercise 16: Brute-force single-byte XOR cipher using the instructions here: https://cryptopals.com/sets/1/challenges/4

Exercise 17: Implement repeating-key XOR using the instructions here: https://cryptopals.com/sets/1/challenges/5