Chapter 3: Set three: Encryption Part II and hashing.
Deadline: Friday October 6, 23:59 (11:59 PM)

Please consult the 'Practical assignments instructions and rules' document before proceeding with the exercises.

Exercise 1. [3 points]
Purpose of this exercise: implement RSA encryption.

Write a program that computes the private RSA key from a public key, then encrypts or decrypts one or more numbers. The input will be as outlined below. The e or d in the first line specifies whether to encrypt or decrypt. The second line consists of values p, q and e, as described during the lecture. This is followed by x lines of numbers to be encrypted. The output should be the encrypted numbers, separated by newlines.

```
[e|d]
p q e
n0
n1
n2
...
```

Note: This exercise will feature intermediate values in the range of 10^{17}. Keep this in mind when deciding what data types to use.

Exercise 2. [3 points]
Purpose: implement a Diffie-Hellman key exchange algorithm featuring Elliptical Curve Cryptography.

For this exercise, you will implement a Diffie-Hellman key exchange based on Elliptical Curve Cryptography. The input of the program will consist of a starting point on the curve $y^2 = x^3 + ax + b$, the values of a, b, and modulo p and lastly the values m and n, as described during the lecture. The structure is as follows:

```
(x, y)
a b p
m n
```

The output of the program should be the shared secret, in the following format:

```
(x', y')
```

Exercise 3. [1.5 points] Purpose: Purpose of this exercise: investigate when a hash collision can be expected.

Assume every person on earth writes a document once a day and signs it using the sha-256 hashing algorithm. Assume the sha256 hashing algorithm is secure.

After (approximately) how much time hash-value collisions must occur with this algorithm?

(state your answers as powers of 10, use the approximation $2^n = 10^{n/3}$

Exercise 4. [1.5 points]
Purpose: Learn to recognize a frequently occurring flaw.

Software is frequently offered by websites for downloading. A nice example is found at https://www.openttd.org/downloads/openttd-releases/latest.html.

On that site it is stated:

> For all binaries officially released by us we publish the MD5, SHA1, and SHA256 checksums. You can use these checksums to check whether the file you downloaded has been modified. All three checksums should match the file you downloaded; if this is not the case it means that either the file didn't come from us or that it got broken during transport. Either way it might possibly contain dangerous modifications and the file should therefore not be trusted!

The checksums are shown after pressing *Checksums*, and the archive may be downloaded next.

Why is the currently used procedure used by `openttd.org` OK? In previous years their download location was `http://www.openttd.org/downloads/openttd-releases/latest.html`. That URI is still available, but it immediately redirects you to the abovementioned web location.

- Is that OK as well?
- Any thoughts about merely using `http://www.openttd.org/downloads/openttd-releases/latest.html`?

Exercise 5. [1 point]
Purpose: obtain some experience with steganography.

Find the hidden message embedded in the first hashing slide Submit the hidden message and describe how you found it.

Hint: there's more to this exercise than meets the eye.

You receive half a point if your answer was basically found by brute force (i.e., trial and error), or even by smart thinking. But if you find the used steganogra-

phy the answer will be immediately clear: in that case report the used steganography (and the hidden message), and you get the full credit point.