Information Security Exercises

Chapter 4: Set four: Hashing. Deadline: October 14, 9:00

NOTE: The deadline has been extended from October 9 to October 14!

Document version: 1.2

You can earn 7 points for these exercises: Exercise 22 is worth 2 points. The other exercises are worth 1 point each.

Notes:

- **Don't forget to submit your source code for all of the programming exercises (both as separate attachment so we can run it, and in the report so we can give feedback).**
- Computing a SHA256 hash is possible using either the built-in cryptography capabililities of your language of choice, or, if it does not have it, by calling into the 'sha256sum' command-line program that is available on Linux systems, `shasum -a 256` on Mac. Another alternative is to call into the Foreign-Function-Interface of OpenSSL. You don't have to implement it yourself.

Exercise 17.
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 18.
Purpose: learn to manipulate CRC computations

Solve exercises 8 and 9 chapter 5, section 5.11; Stamp (2011, p. 155)

Exercise 19.
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/en/download-testing.

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 wheter the file you downloaded got modified. All three checksums should match the file you downloaded; if this is not the case it means that either the file didn't came from us or that it got broken during transport. Either way it might possibly contain dangerous modifications and the file should therefor 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/en/download-testing`. 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/en/download-testing`?

Exercise 20.
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 steganography the answer will be immediately clear: in that case report the used steganography (and the hidden message), and you get the full credit point.

Exercise 21.
Purpose: compute a Hashed Message Authentication Code (HMAC)

The steps of this exercise are as follows:

- Find the zipped document 'ex21.zip' on Nestor.

- Use as HMAC key 'firstname-lastname' where *firstname* and *lastname* are your first and last name, respectively.
- You then compute the (extracted) document's sha256 HMAC hash value, using that HMAC key.
- As answer to this exercise, submit the chosen HMAC key, the computed HMAC hash value, and the source code you used to compute it.

Hint: The OpenSSL library supports HMAC computations through members like `HMAC_CTX_init, HMAC_Init_ex`, and `HMAC_update`.

Exercise 22. *This exercise is worth 2 points*
Purpose of this exercise: investigate what the probability is of obtaining PGP keys with equal fingerprints.

Determine the probability that at least two people (assuming there are 7.5 billion people ($7.5 * 10^9$) on earth) use a PGP key having the same public key fingerprint.

As an illustration, the fingerprint of Frank's GPG key is (in blocks of 4 hexadecimal digits):

DF32 13DE B156 7732 E65E 3B4D 7DB2 A8BE EAE4 D8AA

Hint:

To compute the exact probability is rather difficult, but a very good approximation is possible, where a 'worst case' approach is used. For that approximation the following result can be used:

```
(1 - x) ^ n >= (1 - n * x) for x in [0,1]
```

If you provide a clear explanation and 'educated guess' then you receive a point for completing this exercise.