Exercise / Lab Sheet 1

Data Security for Embedded Systems, DT4020, VT21, LP3

Deadline for submission: **19 February 2021**

The Rules / Introduction

1. The exercises are part of your lab work in the course and should be handed in the prescribed time for grading. Exercises **have to be passed** to get a course grade.
2. The grade for the exercises is **Pass or Fail**.
3. To get a Pass all questions / problems have to be answered / solved to a satisfactory degree, unless noted otherwise (optional exercises).
4. Help is provided during the exercise hours held by Sina, see the schedule. Note that exercise hours may not be evenly distributed over the course weeks.
5. The submission format is **PDF document** with all the answers and the snippets of the code you developed (i.e., a short report). The complete code files should be also submitted in a **separate ZIP file**.
6. Use the Blackboard submission systems (assignments) to submit your solutions.
7. Each exercise sheet can be re-submitted two times (i.e., you have **three attempts** to provide a complete solution). However, the whole process have to be completed before the end of **March 2021**.
8. Requests for any deadline extensions have to be argumented and requested ahead of time (not on the day of the deadline or the day before). An **unannounced late submission** is equivalent to missing one re-submission chance.
9. Any requests or questions should be e-mailed to **Wojciech and Sina** at the same time. **If you send the question to only one of us you risk a delayed answer!**
10. You are allowed to use the help of uncle Google to find your way to the solution, however, direct copying of code or text without acknowledgement (from the Internet, or from other students) **is forbidden** and will have negative **consequences**. Otherwise, any referenced work has to be properly cited.
11. The exercises do not necessarily have one proper solution, there can be many. This is a security course and you are allowed to be or may have to be inventive.

Tasks

1. Go on the web and find information about a recent security problem in IT, which is no older than 2 years and that has not been mentioned during the lectures. Write a couple of paragraphs about the problem, possible consequences and impact, and currently existing solutions (if any) to remedy the problem.

2. Make an attempt at sanitizing and de-obfuscating the mystery C code showed during the first lecture (`test.c` file). Do you know (roughly) what the program does? What approach / methods / programs did you use to get to your conclusion? Give examples of obfuscating methods that were used by the author of the program.

3. In the second lecture we discussed the One-Time-Pad cipher and showed that Eve's guesses for the key are futile as she can get any reasonably looking result. Check that the key she guessed indeed decrypts the ciphertext to "`LATER`". If it does, what should be the key stream to give "`NEVER`" instead. If it does not, correct the key so that you do get "`LATER`". You may do it "pen & paper" style, or write a compute program or spreadsheet to help you out.

4. Develop a `OTPInputStream` in Java that extends the `InputStream` and takes another input stream of key data and provides a stream encrypting / decrypting input stream. Develop a test program to show the use of `OTPInputStream`. Provide two versions (can be a configurable behaviour of the class too) – one using a classical OTP and accepting only the CAPITAL letters in the input and key streams, and one that uses XOR and arbitrary data.

5. Consider the RSA cipher. Assume that the two following prime numbers are selected, P=7 and Q=11. Compute N and W. Select a valid public exponent E. Given the selected public exponent E, compute the private exponent D. What are the public key and private key here? Show an example encryption and decryption with these keys.

6. Lookup the Euclidean algorithm on the web and explain its relation to the RSA calculations, show how it is used in task 5.

7. In Lecture 3 we did RSA in Java using `BigInteger` numbers, in lecture 6 we used the "production" API of Java for RSA. Write a Java program that verifies that this is one and the same thing by using **one method (ours or API) to encrypt** and **the other to decrypt** some message (sequence of bytes from a `BigInteger`). For that you need to generate a full length RSA key (use any method you deem suitable) and take care of the padding.

8. **(Optional)** Try to do the same exercise, but for the DH key agreement, that is verify the functionality of the `BigInteger` version of the algorithm and the API one to be the same. For that you have to find out how the DH JCE API should be used. Do this task as a **`JUnit` test**.

9. Find an SSL protected (HTTPS) website with an invalid certificate (by invalid we mean anything that your web browser is not OK with and complains about / puts an

exclamation mark in the address field), analyse the certificate chain and identify the verification failure reason. How serious is the reason? How did you go about finding such a website?

10. During Lecture 6 you have seen how to encrypt with AES in C using a given key and initialisation vector, and we have seen how to encrypt with AES in Java using Password Based Key Derivation. Do either of the two:

    - Write an AES decryptor in Java that would decrypt data encrypted with `aes-enc.c`, or
    - Write an AES decryptor in C that would decrypt data encrypted with the Java program presented during the lecture (and to be found on Blackboard).

    In both cases provide a test data and example that would show that both programs work with each other.

    Help: Windows users typically have problems compiling the C code without installing a long chain of tools. For this case it is better to choose the first option (develop in Java) and use these test vectors generated by `aes-enc.c`:

    Call: **`./aes-enc 0123456789abcdef 0000000000000000 "Plain text"`**
    Output: **`1ff4ec7cef0e00d81b2d55a4bfdad4ba`**

    Call: **`./aes-enc 0011223344556677 0123456789abcdef "Somewhat longer plain text with non-zero IV"`**
    Output:
    `9e4816cc13810b8424d788fbcd4b006b31bf45f5f9191072820ae0a545500c966cf22afda1002466a78b7e4ddf02587f`