You will need 50% of all homework points to qualify for the exam. (That is, if you get at least 50%, your final grade will be the exam grade. And if you do not get 50%, you do not pass the course.)

You may hand in your solutions in person or by email. If you submit by email, either scan a handwritten solution or typeset your solution readably. I do not consider ASCII formulas readable.

When submitting, indicate your name and your matriculation number. On your first submission, please also indicate a password, this password will be needed for accessing the solutions and your points online.

The total number of points for each homework is 20 (not including points for bonus problems, if available).

For submitting your solution in a nicely typeset way (e.g., using LaTeX), you get up to 3 bonus points, but not more than 30% of the points you reached for content.

Problem 1: One-time-pad

(a) Write a program that achieves the following: It takes as input two ciphertexts \( c_1 \) and \( c_2 \) of the same length. Both are expected to be the encryption of a single word \( m_1, m_2 \) using the one-time-pad. To produce the ciphertexts, the same key has been used. The program then finds \( m_1 \) and \( m_2 \).

Consider the following ciphertexts: \( c_1 = 4A5C45492449552A, c_2 = 5A47534D35525F20 \) (eight bytes each, presented in hex). Figure out the plaintexts using your program.

Note: On many Linux systems, you find a wordlist in /usr/share/dict/words. Or use the file wordlist.txt from the webpage. Please submit a printout of your source code and the plaintexts.

Hint: If you use python (version 3.x), you may find the following code snippets useful: \( \text{bytes.fromhex("5AC643BE8504E35E")} \) decodes a hex string. And the following XORs two string bitwise:

```python
def xor_two_words(x,y):
    assert len(x)==len(y)
    assert isinstance(x,bytes)
    assert isinstance(y,bytes)
    return bytes([a^b for a,b in zip(x,y)])
```
(b) **Bonus problem.** Write a program that does the same as in (a), except that $m_1$, $m_2$ are now English sentences.

This is much more difficult, but if you enjoy the challenge, you can do it.

**Problem 2: Perfect secrecy**

Show that there is no encryption scheme that has perfect secrecy and allows us to reuse the key. More precisely, show that there is no encryption scheme $E$ that satisfies the following definition (and that can be decrypted):

**Definition 1 (Perfect secrecy with key reuse)** Let $K$ be the set of keys, let $M$ be the set of messages, and let $E$ be the encryption algorithm (possibly randomized) of an encryption scheme. We say the encryption scheme has **perfect secrecy with key reuse** iff for all $n$, and all $m_1^{(1)}, \ldots, m_1^{(n)}, m_2^{(1)}, \ldots, m_2^{(n)} \in M$ and for all $c_1, \ldots, c_n$, we have that

$$
\Pr[(c_1, \ldots, c_n) = (c_1', \ldots, c_n') : k \xleftarrow{} K, c_1' \leftarrow E(k, m_1^{(1)}), \ldots, c_n' \leftarrow E(k, m_1^{(n)})] = \Pr[(c_1, \ldots, c_n) = (c_1', \ldots, c_n') : k \xleftarrow{} K, c_1' \leftarrow E(k, m_2^{(1)}), \ldots, c_n' \leftarrow E(k, m_2^{(n)})]
$$

**Hint:** If you have an encryption scheme $E$ with perfect secrecy with key reuse, first construct from it a scheme $E'$ with perfect secrecy that has messages longer than keys. (Show that it indeed has perfect secrecy.) Then use **Theorem 1** in the lecture notes.