Cryptology I (spring 2019) Dominique Unruh

Exercise Sheet 1

Out: 2019-03-02 Due: 2019-03-16

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) In a brute force attack, one tries every possible key k and tries to decrypt the ciphertext c using k. When decrypting c using k yields a valid plaintext (e.g., an English sentence), one has found the key.

Given enough time, one can also enumerate all possible keys for the one-time pad. Thus, given unlimited computational power, one can apply the brute-force attack to the one-time pad. One the other hand, we have proven that the one-time pad has perfect secrecy. Thus is should not be possible to break the one-time pad.

Explain why a brute-force attack fails on the one-time pad (even if one has unlimited time).

(b) 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: `bytes.fromhex("5AC643BE8504E35E")` decodes a hex string. And the following XORs two string bitwise:

```
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)])
```

(c) **[Bonus problem.]** Write a program that does the same as in (b), 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 $m_0^{(1)}, \ldots, m_0^{(n)}, m_1^{(1)}, \ldots, m_1^{(n)} \in M$ and for all $c_1, \ldots, c_n$, we have that*

$$\begin{aligned}&\Pr[(c_1, \ldots, c_n) = (c'_1, \ldots, c'_n) : k \xleftarrow{\$} K, c'_1 \leftarrow E(k, m_0^{(1)}), \ldots, c'_n \leftarrow E(k, m_0^{(n)})] \\ =&\Pr[(c_1, \ldots, c_n) = (c'_1, \ldots, c'_n) : k \xleftarrow{\$} K, c'_1 \leftarrow E(k, m_1^{(1)}), \ldots, c'_n \leftarrow E(k, m_1^{(n)})]\end{aligned}$$

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.