Lab 2: Key Reuse

Karlstad University

Mahdi Akil – mahdi.akil@kau.se
Samuel Wairimu – samuel.wairimu@kau.se

1 Introduction

While a one time pad provides perfect secrecy, neither many-time pads nor stream ciphers with reused key-streams provide any secrecy guarantees. For example, suppose that we have two plaintexts p_1 and p_2, both of which are encrypted using the same key-stream k_{stream}. Since a stream cipher encrypts a plaintext by applying XOR to the key-stream (see Equations 1–2), we get the following unfortunate relation: $p_1 \oplus p_2 = c_1 \oplus c_2$. In other words, by the properties of XOR two equal key-streams cancel each other out.

$$c_1 \leftarrow p_1 \oplus k_{\text{stream}} \tag{1}$$
$$c_2 \leftarrow p_2 \oplus k_{\text{stream}} \tag{2}$$

If you are not convinced that $p_1 \oplus p_2$ leaks any valuable information, please take a look at the graphical example created by Cryptosmith[1].

[1] https://crypto.stackexchange.com/questions/59/taking-advantage-of-one-time-pad-key-reuse, accessed 2022-01-13.

2 Task

Suppose a key-stream reuse vulnerability is found in a wireless protocol. Using Wireshark you captured eleven ciphertexts (see attached file), each reusing the same key-stream. Assuming that the corresponding plaintexts are in English, find a way to *recover the plaintext of the **final** ciphertext*.

You may work alone or in pairs, using any programming language of your choice. Appendix A shows the code used to generate these ciphertexts.

Hint: what happens if a space is XORed with a character in [a-zA-Z]?

3 Submission

Other than successfully completing the task in Section 2, you must write a report that contains the following:

- Full names and email addresses of all group participants.
- A problem description and necessary assumptions.
- Principles and limitations of your exploit code (at least 200 words).
- How to run your code preferably on Linux.
- Encountered issues and challenges (if any).
- The lessons learned by this lab (at least 200 words).

Submit your exploit code (attachment) and the report (PDF) via Canvas by the end of the course. Late labs may not be corrected until the next examination period.

Acknowledgements

The following people have contributed to this lab specification: Mahdi Akil, Rasmus Dahlberg, Leonardo Martucci, Tobias Pulls, Artem Voronkov, and Samuel Wairimu. The lab is based on a similar lab in Cryptography I from Stanford.

A Ciphertext Generation

```
1  #
2  # Source: Cryptography-1 at coursera.org (Stanford University)
3  #
4
5  import sys
6
7  MSGS = ( ---  11 secret messages  --- )
8
9  def strxor(a, b):     # xor two strings of different lengths
10     if len(a) > len(b):
11        return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a[:len(b)], b)])
12     else:
13        return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b[:len(a)])])
14
15 def random(size=16):
16     return open("/dev/urandom").read(size)
17
18 def encrypt(key, msg):
19     c = strxor(key, msg)
20     print
21     print c.encode('hex')
22     return c
23
24 def main():
25     key = random(1024)
26     ciphertexts = [encrypt(key, msg) for msg in MSGS]
```