ECE404 Introduction to Computer Security: Homework 01

Spring 2024
Due Date: 5:59pm, January 18, 2024

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace. You will be able to find the finalized policies regarding the programming assignments and regrade requests on BrightSpace after the first week of class. Please hold off your related questions until then.

1 Introduction

In this exercise, you will assume the role of a cryptanalyst and attempt to break a cryptographic system composed of the two Python scripts `EncryptForFun.py` and `DecryptForFun.py` described in Section 2.11 in Lecture 2. As you will recall, the script `EncryptForFun.py` can be used for encrypting a message file while the script `DecryptForFun.py` recovers that message from the ciphertext produced with the previous script. Both these scripts can be found on the ECE 404 web page for the Lecture Notes (click on the "download code" tab for Lecture 2) [3].

2 Programming Tasks

2.1 Downloading the Necessary Dependencies

Before writing any code, you will first need to download the `BitVector` package [2]. You are free to acquire this dependency in any fashion you wish, however the guidelines below detail the setup via. Anaconda environments.

1. Follow the instructions here [1] for installing Anaconda on your local machine.

2. Create your ece404 conda environment:
   `conda create --name ece404 python=3.8`

3. Activate your new conda environment:
   `conda activate ece404`

4. Install the `BitVector` package:
   `pip install BitVector`

2.2 Problem Statement

With the `BLOCKSIZE` parameter set to 16 and the `passphrase` parameter set to "Hopes and dreams of a million years", the script `EncryptForFun.py` produces the following **one-line** ciphertext for a plaintext message regarding Scuderia Ferrari driver Charles Leclerc.

5e0e392c531926696d58196b3f735c512c640a6469460c737c4216325c07393b5a1
93f695b562f6809093979704a0275664012302559 1a3e3c0749772e0765380d0d6c
3c191d6a284c0a2b3944072577475528734b5161210404702613186124 1e546a65
0f5c676b41522922691d1d206e1c096c40522322431d376d625e116a72481d793b
6f584d2c7c584570353e0018277119573d39081f1f497c6f7f0c5d63681618732014
1d62240b1a7e221102373b5f4b6b2b5c4d7a7c1248346c5a597c4f335e15487c6a1
74f7d7e5b1c7d0f133e781e17217f02113b674b08752e17187629080f146c2706127
221541c332850583e614b110b2e7f130c2f6b5f2261411123244701357 15a4f7c2d4
a4c743c1403607335424670255 20f2268521a677d5a5b6b3b5d54613b5049652b1
55a2c28140e026d2019116d28475827675317222f4d1c236c4d596b7e0c5d2a7d4
d5f3f2f5619703d1e08381e5b3c2f0d5b347d200919792c0d5c4a6f2d5b4e663118

Your job is to recover both the original quote and the encryption key by mounting a brute-force attack on the encryption/decryption algorithms.

HINT 1: The correctly decrypted message should contain the word *Ferrari*.
HINT 2: The logic used in the scripts assumes that the effective key size is 16 bits when the BLOCKSIZE variable is set to 16. So your brute-force attack needs to search through a keyspace of size 2^{16}.

2.3 Programming Instructions

Implement the following function to decrypt the ciphertext within `ciphertextFile` using `key_bv`, which returns the original plaintext as a string.

```
def cryptBreak(ciphertextFile, key_bv):
# Arguments:
# * ciphertextFile: String containing file name of the
                                    ciphertext
# * key_bv: 16-bit BitVector for the decryption key
```

A couple of things to keep note of:

- The function must be implemented and saved in a file named `cryptBreak.py`.
- This function must be implemented to decrypt the message *for a single key* and not to perform complete brute force analysis - the brute force analysis must be done within the code's __main__ function/statement or in a separate Python file by importing `cryptBreak.py` into that file.
- Note that the string returned by the above function may or not may not be the correct plaintext since the correct `key_bv` is unknown. Therefore to determine the correct value for `key_bv`, you will need to brute force all possible values for `key_bv` and check the returned string to find the right one.
- You need to submit only the `cryptBreak.py` file which will be auto-graded - hence make sure that the `cryptBreak.py` file does not run the entire brute force analysis or any other routine when imported.

2.4 Example Usage

Below is an example of how your implemented function could be used - if your function is implemented correctly, the following code snippet should run without any errors.

```
from cryptBreak import cryptBreak
from BitVector import *
RandomInteger = 9999 #Arbitrary integer for creating a BV
key_bv = BitVector(intVal=RandomInteger, size=16)
decryptedMessage = cryptBreak('encrypted.txt', key_bv)
if 'Ferrari' in decryptedMessage:
  print('Encryption Broken!')
else:
  print('Not decrypted yet')
```

3 Submission Instructions

- You must turn in a single zip file on Brightspace with the following naming convention: `HW01_<last_name>_<first_name>.zip`. Do not turn in files other than those listed below. Your submission must include:
  - The file containing your `cryptBreak` implementation named `cryptBreak.py`
  - A pdf named `HW01_<last_name>_<first_name>.pdf` containing:
    * The recovered plaintext quote
    * The recovered encryption key
    * A brief explanation of your code

References

[1] Anaconda Installation Instructions. URL https://conda.io/projects/conda/en/latest/user-guide/install/index.html.

[2] BitVector Python. URL https://pypi.org/project/BitVector/.

[3] ECE 404 Lecture Notes. URL https://engineering.purdue.edu/kak/compsec/Lectures.html.