ECE404 Introduction to Computer Security: Homework 01

Spring 2025
Due Date: 5:59pm, January 23, 2025

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 the Lord of the Rings.

491f2a7e45513073445b326e5d4035215c0e7a3b141f32115a357c2613097264545
23a7a5f5379651e5738705143306752063 63e102c771803366b57047822744b4023
7505706c5e073f600c0938611d476d771c417d2e5208676643405123734a5c283d6
f1b0c7a655d43512c744e592d354e153a20545b72551a713c42536d320c40226611
153c765d0474675841782b550134681b0b7a7f5217743141582038085 02d7618183
c7351557f3d5b1b68744715266708412b35035d257b1015347e551978731555353d
101b3873590170621100317 41b1c3f690d4f2e20014260304953657b24

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 *Sauron*.
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 'Sauron' 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.