ECE404 Introduction to Computer Security: Homework 04

Spring 2024
Due Date: 5:59pm, February 13, 2024

1 Introduction

In our recent lectures, finite fields have been a key focus. These mathematical structures lay the groundwork for the Advanced Encryption Standard (AES), enabling the secure handling of arithmetic operations within a limited set. This understanding will be crucial as we embark on implementing the AES algorithm, where finite fields are at the heart of the encryption process.

As always, please read the homework document in its entirety before coming to office hours with your questions. The teaching staff have spent a long time writing the assignment to cover many common questions you might have.

2 Programming Assignment

Write an Object Oriented Python [1] program that implements the full AES algorithm. More specifically, given a **256-bit key** and a plaintext message, your program must produce the correct encryption and decryption results.

The two commands below specify the exact command-line syntax for invoking encryption and decryption.

```
1 python3 AES.py -e message.txt key.txt encrypted.txt
2 python3 AES.py -d encrypted.txt key.txt decrypted.txt
```

An explanation of the command-line syntax is as follows:

- Encryption (indicated with the -e argument in line 1)
  - perform AES encryption on the plaintext in `message.txt` using the key in `key.txt`, and write the ciphertext to a file called `encrypted.txt`
  - You can assume that `message.txt` and `key.txt` contain **text strings** (i.e. ASCII characters)
  - However, the final ciphertext should be saved as a single-line **hex string**

- Decryption (indicated with the -d argument in line 2)
  - perform AES decryption on the ciphertext in `encrypted.txt` using the key in `key.txt`, and write the recovered plaintext to `decrypted.txt`

A skeleton file for your `AES.py` has been provided below.

```python
import sys
from BitVector import *

class AES():
    # class constructor - when creating an AES object, the
    # class's constructor is executed and instance variables
    # are initialized
    def __init__(self, keyfile:str) -> None:

    # encrypt - method performs AES encryption on the plaintext and writes the ciphertext to disk
    # Inputs: plaintext (str) - filename containing plaintext
    #         ciphertext (str) - filename containing ciphertext
    # Return: void
    def encrypt(self, plaintext:str, ciphertext:str) -> None:

    # decrypt - method performs AES decryption on the ciphertext and writes the recovered plaintext to disk
    # Inputs: ciphertext (str) - filename containing ciphertext
    #         decrypted (str) - filename containing recovered plaintext
    # Return: void
    def decrypt(self, ciphertext:str, decrypted:str) -> None:

if __name__ == "__main__":
    cipher = AES(keyfile = sys.argv[3])

    if sys.argv[1] == "-e":
        cipher.encrypt(plaintext=sys.argv[2], ciphertext=sys.argv[4])
    elif sys.argv[1] == "-d":
        cipher.decrypt(ciphertext=sys.argv[2], decrypted=sys.argv[4])
    else:
        sys.exit("Incorrect Command-Line Syntax")
```

2.1 Useful Notes For AES implemenation

The following points may aid you in your implementation of AES, however for full documentation, please refer to Lecture 8 [3].

- Each round of AES involves the following four steps
  1. Single-byte based substitution
  2. Row-wise permutation
  3. Column-wise mixing
  4. XOR with the round key
- Please note that the order in which these four steps are executed is different for encryption and decryption.
- The last round for encryption does not involve the 'Mix Columns" Step. Similarly, the last round for decryption does not involve the 'Inverse Mix Columns" step.
- As you know, AES has variable key-length, and the number of rounds of processing depend upon the key-length. The lecture assumes a 128-bit key length and all subsequent explanation is based upon that assumption. But the key provided to you is 256 bits long, hence, there will be a slight variation in how you generate the *key schedule*. The following explanation will be helpful in that regard:
  1. For the key expansion algorithm, note that irrespective of the key-length, each round still uses only 4 words from the *key schedule*. Just as we organised the 128-bit key in 4 words for key-expansion, we organise the 256-bit key in 8 words.
  2. Each step of the key-expansion algorithm takes us from 8 words in one round to 8 words in the next round. Hence, 8 such steps will give us a 64-word key schedule. The implementation of the $g(\cdot)$ function remains the same. The logic of obtaining the 8 words from the j^{th} step of key expansion to the $(j+1)^{th}$ step also remains the same.
  3. Note that since the key is 256-bits long, there will be 14 rounds of processing in the AES, plus the initial processing. Because each round of processing uses only 4 words from the *key schedule*, you will require only a 60-word *key schedule*. However, the previous step generates a 64-word schedule, so you can ignore the last 4 words in the schedule.

- Keep in mind that the block size is still 128 bits, despite the key size being 256 bits.
- Should the last block of plaintext not be an integral multiple of 128 bits, pad the block with trailing 0's. Note that this will result in trailing NULL bytes in your recovered plaintext files.
- We have provided `first_round.txt` which allows you to verify your results for each of the four steps in the **first round of processing the first block**.
- You can verify your final ciphertext with this online tool from javainuse [2]. Please note that due to different padding methods for blocks that are not integral multiples of the block size, the final block of encryption generated from the website will not match with your final encrypted block.

3 Submission Instructions

Make sure that the program requirements and submission instructions are followed. Failure to follow these instructions may result in loss of points!

- For this homework you will be submitting a zip file to Brightspace titled `hw04_<last_name>_<first_name>.zip` containing:
  - The file `AES.py` containing your code for the programming problem.
  - A PDF titled `hw04_<last_name>_<first_name>.pdf` containing:
    * a brief explanation of your code, and the encrypted and decrypted output for the text mentioned above using the key provided.

References

[1] Object-Oriented Programming in Python. URL https://realpython.com/python3-object-oriented-programming/.

[2] AES Encryption and Decryption Online Tool. URL https://www.javainuse.com/aesgenerator.

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