ECE 404 Homework 4

Elias Talcott

February 18, 2020

Contents

1 Theory Problems

1.1 Problem 1

Determine the following in GF(11):

(a) $(3x^4 + 5x^2 + 10) - (8x^4 + 5x^2 + 2x + 1)$

(b) $(5x^2 + 2x + 7) \cdot (5x^3 + 3x^2 + 3x + 2)$

(c) $\dfrac{x^5 + 8x^4 + x^3 + 4x^2 + 8x}{6x^3 + 3x^2 + 2}$

Solution

(a)

(b)

(c)

1.2 Problem 2

For the finite field GF(2^3), calculate the following for the modulus polynomial $x^3 + x^2 + 1$.

(a) $(x^2 + x + 1) \cdot (x + 1)$

(b) $(x^2 + 1) - (x^2 + x + 1)$

(c) $\dfrac{x^2 + x + 1}{x^2 + 1}$

Solution

(a)

(b)

(c)

2 Programming Problem

Write a script in Python to implement the AES algorithm with a 256-bit key size.

2.1 Python Code

```
#!/usr/bin/env python3

# Homework Number: 4
# Name: Elias Talcott
# ECN Login: etalcott
# Due Date: February 18, 2020

###
## Encryption call: python3 AES.py -e message.txt key.txt encrypted.txt
## Decryption call: python3 AES.py -d encrypted.txt key.txt decrypted.txt
###

import sys
from BitVector import *

###
## Encryption algorithm
###
def encrypt(infile, keyfile, outfile):
    # Initialize block size
    BLOCKSIZE = 128

    # Create bitvectors for plaintext, key, and ciphertext
    with open(infile, "r") as fpin:
        plaintext_bv = BitVector(textstring = fpin.read())
    ciphertext_bv = BitVector(size = 0)
    with open(keyfile, "r") as fpkey:
        key_text = fpkey.read()
    if len(key_text) != 32:
        sys.exit("Key generation needs 32 characters exactly!")
    key_bv = BitVector(textstring = key_text)

    # Generate round keys


    # Encrypt plaintext
    plaintext_bv.pad_from_right(BLOCKSIZE - (len(plaintext_bv) % BLOCKSIZE))
    numblocks = len(plaintext_bv) // BLOCKSIZE
    for i in range(numblocks):
        bv = plaintext_bv[i * BLOCKSIZE:(i + 1) * BLOCKSIZE]
```

```
    # Save ciphertext to output file
    with open(outfile, "w") as fpout:
        fpout.write(ciphertext_bv.get_bitvector_in_hex())


###
## Decryption algorithm
###
def decrypt(infile, keyfile, outfile):
    # Initialize block size
    BLOCKSIZE = 128

    # Create bitvectors for the plaintext, ciphertext, and key
    with open(infile, "r") as fpin:
        ciphertext_bv = BitVector(hexstring = fpin.read())
    plaintext_bv = BitVector(size = 0)
    with open(keyfile, "r") as fpkey:
        key_text = fpkey.read()
    if len(key_text) != 32:
        sys.exit("Key generation needs 32 characters exactly!")
    key_bv = BitVector(textstring = key_text)

    # Generate round keys


    # Decrypt ciphertext
    numblocks = len(ciphertext_bv) // BLOCKSIZE
    for i in range(numblocks):
        bv = ciphertext_bv[i * BLOCKSIZE:(i + 1) * BLOCKSIZE]

    # Save plaintext to output file
    with open(outfile, "w") as fpout:
        fpout.write(plaintext_bv.get_text_from_bitvector())


###
## Check arguments and choose encrypt or decrypt option
###
if len(sys.argv) != 5:
    sys.exit("Wrong arguments!")

if sys.argv[1] == "-e":
    encrypt(sys.argv[2], sys.argv[3], sys.argv[4])
elif sys.argv[1] == "-d":
    decrypt(sys.argv[2], sys.argv[3], sys.argv[4])
else:
    sys.exit("Wrong arguments!")
```

2.2 Code Explanation