Thomas Mackintosh
Box #219

ECE 4802

15 November 2016

Assignment #3

1. Compute in GF(2^8): ($x^6 + x^5 + x + 1$)/($x^7 + x^6 + x^4 + 1$) where the irreducible polynomial is the one used by AES, P(x) = $x^8 + x^4 + x^3 + x + 1$.

In *GF(2^8)*: $x^7 + x^6 + x^4 + 1 \underset{bin}{\Longrightarrow} (1101\ 0001)_2 \underset{hex}{\Longrightarrow} (D1)_{16}$

From the table of multiplicative inverses in *GF(2^8)* we get: $(D1)_{16} \Rightarrow (07)_{16}$

$(07)_{16}$ represented in binary is $(0000\ 0111)_2$. This gives us the inverse polynomial of $x^2+x=1$.

We can now perform multiplication instead of division:

$$(x^6 + x^5 + x + 1) * (x^2 + x + 1)$$
$$= x^8 + 2x^7 + 2x^6 + x^5 + x^3 + 2x^2 + 2x + 1$$

Since the degree of the multiplication is equal to the degree of the irreducible polynomial *P(x)* we must find the modulo:

$$\begin{array}{r|l} & 1 \\ x^8 + x^4 + x^3 + x + 1 & x^8 + 2x^7 + 2x^6 + x^5 + x^3 + 2x^2 + 2x + 1 \\ & x^8 \qquad\qquad x^4 + x^3 \qquad\qquad x + 1 \\ \hline & 2x^7 + 2x^6 + x^5 + x^4 + 2x^2 + \square x \end{array}$$

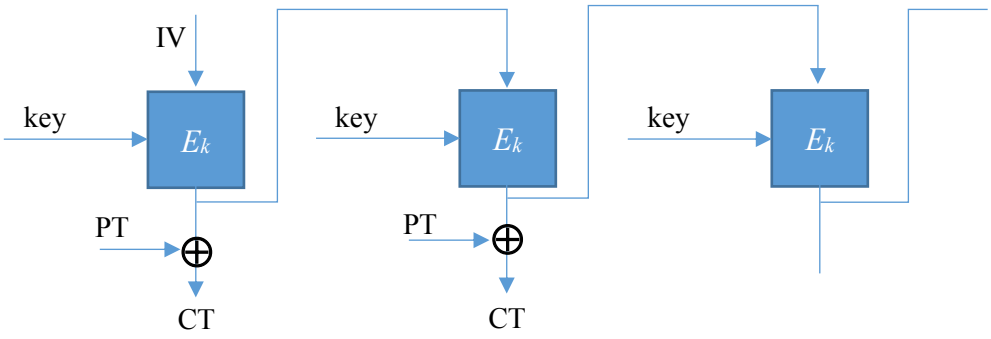
Therefore, the result of $(x^6 + x^5 + x + 1)/(x^7 + x^6 + x^4 + 1) = x^7 + x^6 + x^5 + x^4 + x^2 \square + x$

2. Consider a modified substitution-permutation network where instead of carrying out the key-mixing, substitution, and permutation steps in alternating order for r rounds, the cipher instead first applies r rounds of key-mixing, then carries out r rounds of substitution, and finally applies r permutations. Analyze the security of this construction.

If we perform the key rounds all in a row, the key itself ends up being the total sum of all the rounds of key-mixing. This is equivalent to the key going through only one round of key-mixing instead of the *n* rounds it actually went through. This makes the algorithm now very weak as the key could potentially be derived by XOR'ing two messages together.

3. **As is often true in cryptography, it is easy to weaken a seemingly strong scheme by small modifications. Assume a variant of the OFB mode by which we only feed back the 8 most significant bits of the cipher output. We use AES and fill the remaining 120 input bits to the cipher with zeros.**

(a) Draw a block diagram of the scheme.



(b) Why is this scheme weak if we encrypt moderately large blocks of plaintext, say 100 kByte? What is the maximum number of known plaintexts an attacker needs to completely break the scheme?

Since we know 120 bits of the input to the are zeros we will just need to observe how the key influences those 8 bits. Thus we should only need to test 2^8 key combinations which is very easily computed. With large blocks of plaintext we get a larger amount of blocks to compare so we can identify patterns to produce the key.

(c) Let the feedback byte be denoted by FB. Does the scheme become cryptographically stronger if we feedback the 128-bit value FB, FB,…,FB to the input (i.e., we copy the feedback byte 16 times and use it as AES input)?

No, the stream does not become strong. Since we are still only using the first byte for feedback and since we know what the padding is, we know the padding is essentially meaningless for the rest of the computation as long as we only use a single byte as the basis of the input/output.

(d) Replace the original zero padding of the encryption scheme described above with a new padding scheme that restores the secrecy requirement. Which essential property do you need to add to the encryption scheme?

The encryption scheme must use the full 128 bit output of the previous block as its input. This gets rid of the predictability of the input/output and brings the key space back to 2^{128} which is very difficult to brute force.

4. **The goal of this problem is to encrypt the payload of a .bmp file using three different modes of operation. The cipher to be used is AES. Please write code to encrypt the payload of Gompei.bmp using AES in (i) electronic codebook (ECB) mode (ii) cipher block chaining (CBC) mode and (iii) counter (CTR) mode of operation.**

The code on the next page encrypts the .bmp in ECB mode, CBC mode, and CTR mode all at once:

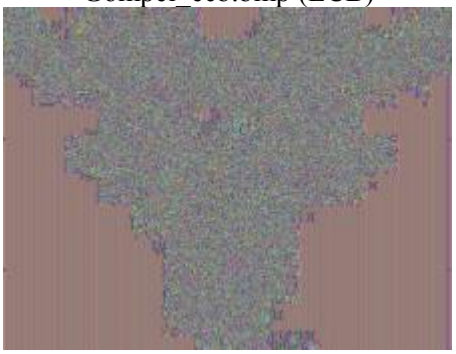
```
  # Template for Project 3:
   # Name: Thomas Mackintosh
   # Date: 12 November 2016
   # Mailbox number: 219

   # This is a simple Python (2) or sage program to open, read and write a .bmp file.
   # Your task is to modify it so that the payload (but not the header) of
   # the .bmp file is encrypted.

from Crypto.Cipher import AES
from Crypto.Util import Counter

for i in range(0,3): #Run through 3 times to producte ECB, CBC and CTR bmp
    if i == 0: mode = AES.MODE_ECB; tag = 'ecb'
    if i == 1: mode = AES.MODE_CBC; tag = 'cbc'
    if i == 2: mode = AES.MODE_CTR; tag = 'ctr'

    # opening the source file (a .bmp file)
    with open('Gompei.bmp','rb') as in_file:
        # Playing a bit with the .BMP file format:

        # get size info:
        in_file.seek(2)
        # size = int.from_bytes(in_file.read(4),'little') # python 3
        size = int(in_file.read(4)[::-1].encode('hex'),16) # [::-1] fixes the endianness
        print('The file has ',size,' bytes.')

        # get to where the bit map begins
        in_file.seek(10)
        # offset = int.from_bytes(in_file.read(4),'little')
        offset = int(in_file.read(4)[::-1].encode('hex'),16) # [::-1] fixes the endianness
        print('Data starts at: ',offset)

        ## read data in chunks, encrypt and write to new file
        #open new file for writing
        with open('Gompei_'+tag+'.bmp','wb') as out_file:

            # read/write header
            in_file.seek(0)
            header = in_file.read(offset)
            out_file.write(header)

            #** Insert your code to encrypt the data ********
            #** using the appropriate mode of encryption ****
            #** below here:

            ####################################
            ####ENCRYPTION STARTS HERE##########
            ####################################

            key = "0000000000000000"
            iv = "0000000000000000"


            if mode == AES.MODE_CTR:
                ctr = Counter.new(128)
                cipher = AES.new(key, mode, iv, counter = ctr)

            else: cipher = AES.new(key, mode, iv)
            # read data in chunks of 64 bit
            buf = in_file.read(16)


            while len(buf)==16:
                # write data in chunks of 64 bit:
                out_file.write(cipher.encrypt(buf))

                # read data in chunks
                buf = in_file.read(16)


            #** Insert your code to encrypt the data ********
            #** using the appropriate mode of encryption ****
            #** above this point

            # write final chunk of data (not necessary since size is multiple of 16)
            out_file.write(cipher.encrypt(buf))
            print 'Successfully output Gompei_'+tag+'.bmp'
        # files are automatically closed after leaving the "with"
```

Following is the console output and the resulting images:

```
('The file has ', 2365578, ' bytes.')
('Data starts at: ', 138)
Successfully output Gompei_ecb.bmp
('The file has ', 2365578, ' bytes.')
('Data starts at: ', 138)
Successfully output Gompei_cbc.bmp
('The file has ', 2365578, ' bytes.')
('Data starts at: ', 138)
Successfully output Gompei_ctr.bmp
```

Gompei.bmp (ORIGINAL)

Gompei_ecb.bmp (ECB)



Gompei_cbc.bmp (CBC)

Gompei_ctr.bmp (CTR)