Practice 3

Finite fields

GF(2^n) and AES key schedule (128 / 192 bits)

Session lab 3 | Friday, April 3, 2020

By Joel Harim Hernández Javier

Description

In this session we will work with finite fields. Please do the following programming exercises on your own. Use only one programming language (C/C++, Java, Python). It is recommended that you try to solve the exercises on your own. You can discuss the solution to the exercises with your colleagues but you should not copy source code. If copying is detected, that may immediately lead to a grade less than 6.

Programming Exercises

1. Design a function that receives $3 \le n \le 8$ for $GF(2^n)$ and as outputs the multiplication table. Consider the following requirements.
    a. The user can choose to represent each element in $GF(2^n)$ as a polynomial or as an hexadecimal number.
    b. The output must be stored in a file.

2. Implement the key schedule for AES, considering a key size of 128 bits. Your program must receive the key in hexadecimal and must store the 10 subkeys (derived from the key) in a file.
3. Repeat the previous point, but now for a key size of 192 bits.

Products

- ➔ The most important parts of the source code.
- ➔ Include screen capture of your programs showing how they work.
- ➔ Please write a small user manual to know how to run your programs.

Table generation

Polynomial multiplied with x^1

In `table.hpp`

```
// Multiplies a polynomial 'a' with x^1 in its binary
// representation modulated 'm', under the field 'n'
inline
short multByX(short a, short m, int n){

    m &= ~((-1) << n);

    return (a & (1 << (n - 1))) ? ((a << 1) ^ m) : (a << 1);

}
```

Polynomial multiplied with x^times

In `table.hpp`

```
// Multiplies a polinomial 'a' with x^times in its binary
// representation modulated 'm', under the field 'n'
inline
short multByManyX(short a, short m, int n, int times){

    if (times == 0) return a;
    if (times == 1) return multByX(a, m, n);

    for (int i = 0; i < times; ++i)
        a = multByX(a, m, n);

    return a;

}
```

Multiply two polynomials module m

In `table.hpp`

```
// Multiplies the polynomials a and b modulated m
// under the field n
inline
short mult(short a, short b, short m, int n){

    bool flag = false;
    short result = 0;

    for (int i = 0; i < n; ++i){

        if (b & (1 << i)){

            result = !flag ?
                  multByManyX(a, m, n, i) : result ^ multByManyX(a, m, n, i);
            flag = true;

        }

    }

    return result;

}
```

Table creation

In `table.hpp > table > writeTable()`

```
    // Table creation
    for (int i = 1; i < (1 << n); ++i){
        for (int j = 1; j < (1 << n); ++j){
            bin ?
                  (writer << bits(mult(i, j, m, n), n))
                : (writer << hex(mult(i, j, m, n), n));
            writer << " ";
        }
        writer << "\n";
    }
```

AES Key Schedule

The last word rotation (first step)

In `aes.hpp`

```
// Rotate the given word
inline
void shiftColumn(unsigned int & column){

    auto aux = column >> 24;

    column = column << 8 | aux;

}
```

The sbox substitution (second step)

In `aes.hpp`

```
// Substitute the values of the column with the sbox's values
inline
void sBoxSubs(unsigned int & column){

    unsigned char * aux = (unsigned char *)(&column);

    for (int i = 0; i < 4; ++i)
        aux[i] = sBox[aux[i] >> 4][aux[i] & 0xF];

}
```

XOR with the round coefficient (third step)

In `aes.hpp`

```
// XOR with the round coeffient
inline
void roundCoeffiecient(unsigned int & column, int i){

    column ^= rc[i] << 24;

}
```

XOR with all the columns (last step)

In `aes.hpp`

```
// XOR with the rest of the columns
inline
void finalBuild(std::vector<unsigned int> & key, unsigned int column){

    for (int i = 0; i < key.size(); ++i){

        key[i] ^= column;
        column = key[i];

    }

}
```

AES key schedule round

In `aes.hpp`

```
// AES key schedule round
inline
void round(std::vector<unsigned int> & key, int i){

    unsigned int column = key[key.size() - 1];

    shiftColumn(column);
    sBoxSubs(column);
    roundCoeffiecient(column, i);
    finalBuild(key, column);

}
```

AES key schedule

In `aes.hpp`

```
// The key schedule process. Receives a string with the key's hexadecimal representation
// If the given key is 128 bits length (32 characters), executes 10 rounds
// If the given key is 192 bits length (48 characters), executes 8 truncated rounds
// Returns a integer vector with all words generated (44 if 128, 52 if 192).
inline
auto keySchedule(std::string keyString){

    if (keyString.size() != 32 && keyString.size() != 48)
        return std::vector<unsigned int>(0);

    auto key = getKeyFromStr(keyString);

    std::vector<unsigned int> result(key.size() == 4 ? 44 : 52);

    for (int i = 0; i < result.size(); i += key.size()){

        for (int j = 0; j < key.size() && i + j < result.size() ; ++j)
            result[i + j] = key[j];

        // There is no point in making a round that will not be used
        if (i + key.size() >= result.size())
            break;

        round(key, i / key.size());

    }

    return result;

}
```

AES Key Schedule pseudocode

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Note that *Nk*=4, 6, and 8 do not all have to be implemented; they are all included in the conditional statement above for conciseness. Specific implementation requirements for the Cipher Key are presented in Sec. 6.1.

Figure 11. Pseudo Code for Key Expansion.[2]

Instructions

Compilation

Just use the command make:

```
make
```

or use the compiler:

```
g++ ./src/p3 -o ./build/p3
```

And change to the build folder, where are some input examples:

```
cd build
```

Usage

The program must be used as follows:

```
./p3 < in.txt
```

The `in.txt` must be have the following structure:

An integer $O,\ 0 \leq O \leq 1$, that selects between the table creation (0) or the aes schedule (1).

If the option 0 (table creation) is selected, the next line must be a integer $n,\ 3 \leq n \leq 8$, the order of the Galois Field, and the next line a string: bin or hex, the way of output the numbers. The result will be in `table.txt`

If the option 1 (aes schedule) is selected, the nex line must be a string with a valid hexadecimal representation of a 128 bits key or 192 bits key, without spaces and with 0 values as 00 (in order to have 32 or 48 characters respectively).

Examples

Example 1

`in.txt`

```
0
3
bin
```

`table.txt`

```
m(x): 1 0 1 1
0 0 1  0 1 0  0 1 1  1 0 0  1 0 1  1 1 0  1 1 1
```

```
0 1 0  1 0 0  1 1 0  0 1 1  0 0 1  1 1 1  1 0 1
0 1 1  1 1 0  1 0 1  1 1 1  1 0 0  0 0 1  0 1 0
1 0 0  0 1 1  1 1 1  1 1 0  0 1 0  1 0 1  0 0 1
1 0 1  0 0 1  1 0 0  0 1 0  1 1 1  0 1 1  1 1 0
1 1 0  1 1 1  0 0 1  1 0 1  0 1 1  0 1 0  1 0 0
1 1 1  1 0 1  0 1 0  0 0 1  1 1 0  1 0 0  0 1 1
```

Example 2

in.txt

```
0
4
hex
```

table.txt

```
m(x): 13
1 2 3 4 5 6 7 8 9 a b c d e f
2 4 6 8 a c e 3 1 7 5 b 9 f d
3 6 5 c f a 9 b 8 d e 7 4 1 2
4 8 c 3 7 b f 6 2 e a 5 1 d 9
5 a f 7 2 d 8 e b 4 1 9 c 3 6
6 c a b d 7 1 5 3 9 f e 8 2 4
7 e 9 f 8 1 6 d a 3 4 2 5 c b
8 3 b 6 e 5 d c 4 f 7 a 2 9 1
9 1 8 2 b 3 a 4 d 5 c 6 f 7 e
a 7 d e 4 9 3 f 5 8 2 1 b 6 c
b 5 e a 1 f 4 7 c 2 9 d 6 8 3
c b 7 5 9 e 2 a 6 1 d f 3 4 8
d 9 4 1 c 8 5 2 f b 6 3 e a 7
e f 1 d 3 2 c 9 7 6 8 4 a b 5
f d 2 9 6 4 b 1 e c 3 8 7 5 a
```

Example 3

in.txt

```
1
a000000000000000000000000000b000
```

key.txt

```
Round: 0
A0 00 00 00
00 00 00 00
```

```
00 00 00 B0
00 00 00 00
Round: 1
C2 C2 C2 C2
E7 E7 E7 E7
63 63 63 D3
63 63 63 63
Round: 2
54 96 54 96
81 66 81 66
98 FB 98 4B
46 25 46 25
Round: 3
63 F5 A1 37
32 54 D5 B3
A7 5C C4 8F
D6 F3 B5 90
Round: 4
06 F3 52 65
41 15 C0 73
C7 9B 5F D0
4C BF 0A 9A
Round: 5
99 6A 38 5D
31 24 E4 97
7F E4 BB 6B
01 BE B4 2E
Round: 6
31 5B 63 3E
4E 6A 8E 19
4E AA 11 7A
4D F3 47 69
Round: 7
A5 FE 9D A3
94 FE 70 69
B7 1D 0C 76
FF 0C 4B 22
Round: 8
DC 22 BF 1C
AC 52 22 4B
24 39 35 43
F5 F9 B2 90
Round: 9
74 56 E9 F5
B6 E4 C6 8D
44 7D 48 0B
69 90 22 B2
Round: 10
1F 49 A0 55
9D 79 BF 32
73 0E 46 4D
8F 1F 3D 8F
```

Example 4

in.txt

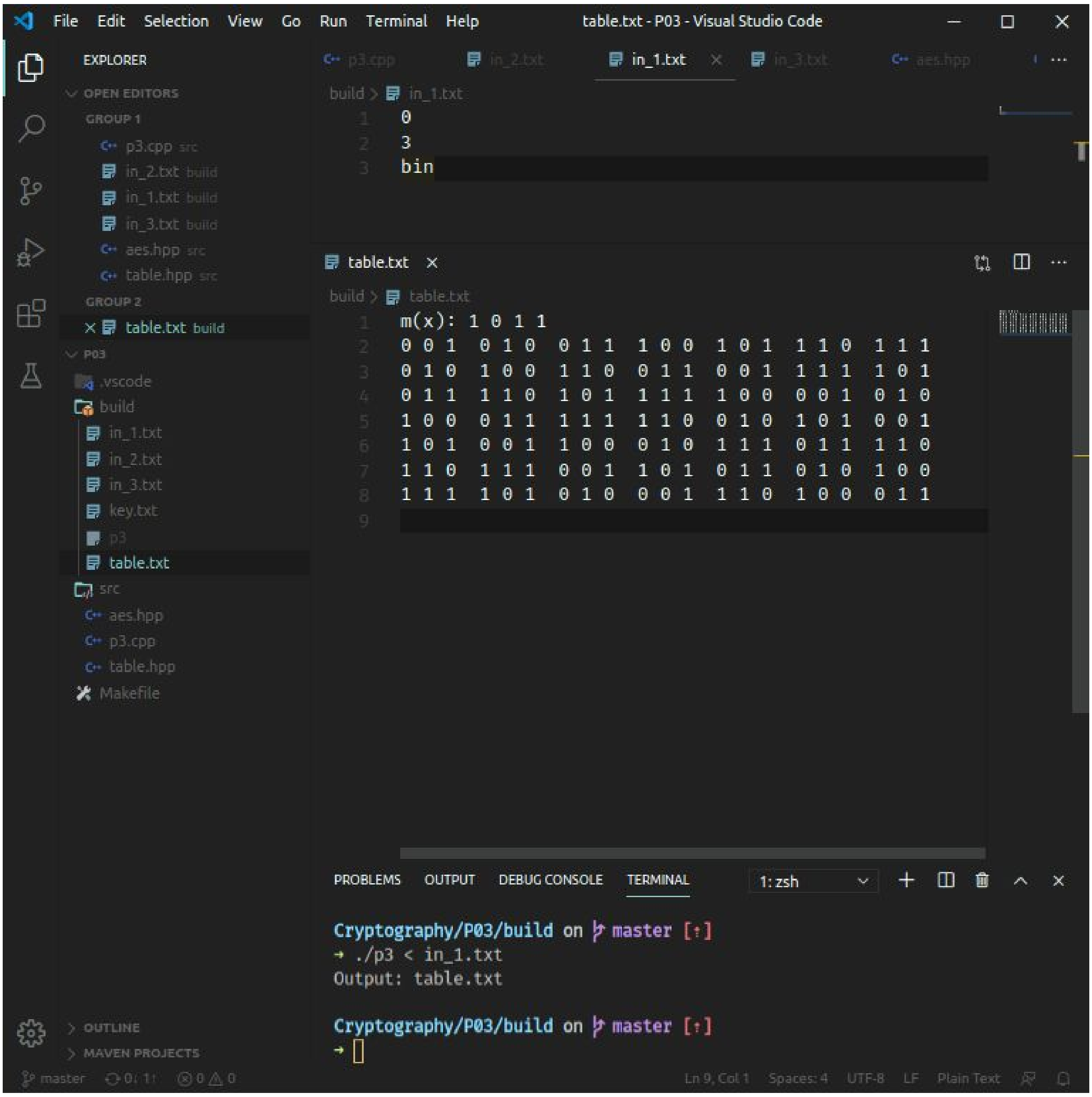
```
1
8e73b0f7da0e6452c810f32b809079e562f8ead2522c6b7b
```

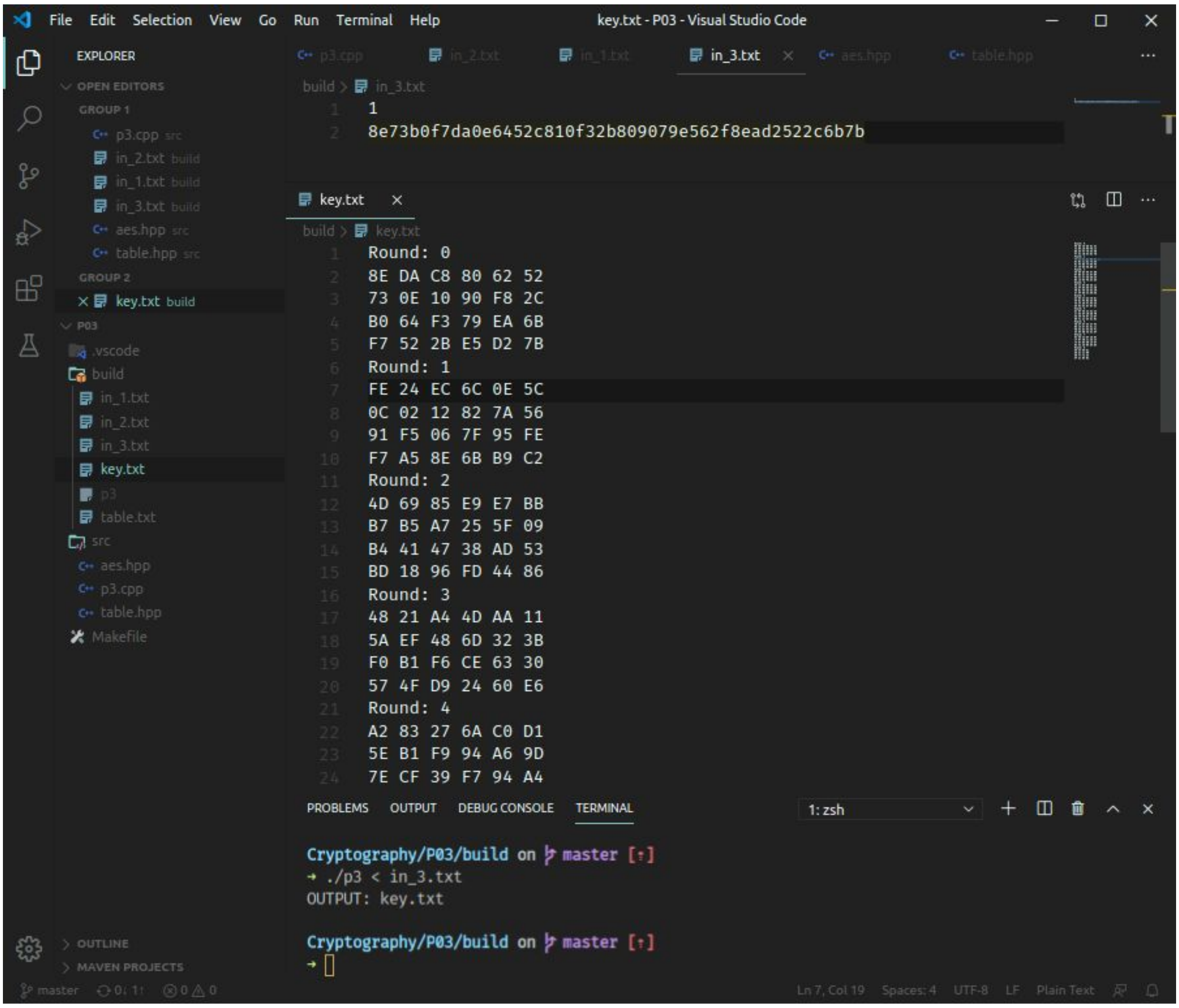
key.txt

```
Round: 0
8E DA C8 80 62 52
73 0E 10 90 F8 2C
B0 64 F3 79 EA 6B
F7 52 2B E5 D2 7B
Round: 1
FE 24 EC 6C 0E 5C
0C 02 12 82 7A 56
91 F5 06 7F 95 FE
F7 A5 8E 6B B9 C2
Round: 2
4D 69 85 E9 E7 BB
B7 B5 A7 25 5F 09
B4 41 47 38 AD 53
BD 18 96 FD 44 86
Round: 3
48 21 A4 4D AA 11
5A EF 48 6D 32 3B
F0 B1 F6 CE 63 30
57 4F D9 24 60 E6
Round: 4
A2 83 27 6A C0 D1
5E B1 F9 94 A6 9D
7E CF 39 F7 94 A4
D5 9A 43 67 07 E1
Round: 5
EC 6F 48 22 E2 33
17 A6 5F CB 6D F0
86 49 70 87 13 B7
EB 71 32 55 52 B3
Round: 6
40 2F 67 45 A7 94
BE 18 47 8C E1 11
EB A2 D2 55 46 F1
28 59 6B 3E 6C DF
Round: 7
82 AD CA 8F 28 BC
1F 07 40 CC 2D 3C
75 D7 05 50 16 E7
0A 53 38 06 6A B5
Round: 8
E9 44 8E 01
```

```
8B 8C CC 00
A0 77 72 22
6F 3C 04 02
```

Screenshots





The complete source code can be found at my personal Github 🙂
github.com/JoelHernandez343/Cryptography
And inside the .zip file in the drive folder.