Fall 2021

Section 1: TTh 3:30pm - 4:45pm - JFSB B092 (changed from MARB 130)

Project #1: AES

Objectives

Gain a concrete understanding of the Advanced Encryption Standard (AES) by implementing it for **128, 192, and 256** bit keys.

Gain experience implementing a system from a specification. You may find some areas of the specification difficult to understand. We encourage you to study the specification carefully. Discuss it with other students, the TA, or the instructor. Be tenacious. You will need to study the language-independent specification and implement the standard in your chosen operating system and programming language. You will have to deal with factors such as byte order, signed and unsigned datatypes, etc.

Resources

- FIPS Publication 197 is the official standard for AES. Each section in the requirements below will reference the appropriate section in this document.
- Stick figure guide to AES
- Wikipedia has a nice overview of AES
- animation of AES on YouTube
- lecture slides on AES.

Implementation Requirements

1. The AES algorithm makes use of finite field arithmetic. As such, it is important to read and understand Section 4. In particular you should implement:

   `ffAdd()` - adds two finite fields (see Section 4.1)

   `xtime()` - multiplies a finite field by x (see Section 4.2.1)

   `ffMultiply()` - uses xtime to multiply any finite field by any other finite field. (see Section 4.2.1)

2. Implement Key Expansion. Notice that the key for use with this algorithm is given as a byte[], but both cipher and invCipher require a word[] as input. The key expansion algorithm (see Section 5.2) performs this conversion. Appendix A gives excellent examples of the Key expansion algorithm. The following two functions are needed by this algorithm:

   `subWord()` - takes a four-byte input word and substitutes each byte in that word with its appropriate value from the S-Box. The S-box is provided (see Section 5.1.1).

   `rotWord()` - performs a cyclic permutation on its input word.

3. Implement the `cipher` function. The cipher function is specified in Section 5.1, and an example is given in appendix B. Its implementation is quite simple once the following four transformations are created:

   `subBytes()` - This transformation substitutes each byte in the State with its corresponding value from the S-Box.

   `shiftRows()` - This transformation performs a circular shift on each row in the State (see Section 5.1.2)

   `mixColumns()` - This transformation treats each column in state as a four-term polynomial. This polynomial is multiplied (modulo another polynomial) by a fixed polynomial with coefficients (see Sections 4.3 and 5.1.3).

   `addRoundKey()` - This transformation adds a round key to the State using XOR.

4. Implement the `invCipher` function. This function is specified in Section 5.3. It reverses the effect of the cipher function. Its implementation is quite simple once the following three transformations are created:

`invSubBytes()` - This transformation substitutes each byte in the State with its corresponding value from the inverse S-Box, thus reversing the effect of a subBytes() operation.

`invShiftRows()` - This transformation performs the inverse of shiftRows() on each row in the State (see Section 5.3.1)

`invMixColumns()` - This transformation is the inverse of mixColumns (see Section 5.3.3).

Submission Requirements

1. Programming language: You may implement your solution for this project in any language in which you are confident you can implement all necessary functions and can demonstrate portions of your solution to the TA using a debugger.
2. To receive full credit, you must implement the system using only the resources listed above. You must avoid looking at any AES source code, which will limit your learning experience.
3. Your code should correctly encrypt and decrypt ALL of the test cases in Appendix C of the FIPS specification. Additionally, you should print out the debug information for each round as seen in Appendix C.

Submission

1. Submit a file with your source code to Learning Suite. **You can use zip or gzip, but please be sure that when the files are uncompressed they are created inside of a new directory, not in the current directory.** For example, they can be inside of a directory called `project`.
2. Include a README file that includes the following:
    - how to compile and run your code
    - a statement about whether you looked at only the resources listed above or whether you used additional material (10% penalty)
    - a statement about which test cases in Appendix C you pass or fail
3. Use the auto grader at https://grader.cs465.byu.edu/aes to pass off your code. Include a screenshot of the success page or partial completion inside your zip file

Some helpful arrays

Key expansion also uses the round constant word array. This array can be generated using `xtime()`, but is provided here for your convenience.

```
// Rcon[] is 1-based, so the first entry is just a place holder
Rcon[] = { 0x00000000,
           0x01000000, 0x02000000, 0x04000000, 0x08000000,
           0x10000000, 0x20000000, 0x40000000, 0x80000000,
           0x1B000000, 0x36000000, 0x6C000000, 0xD8000000,
           0xAB000000, 0x4D000000, 0x9A000000, 0x2F000000,
           0x5E000000, 0xBC000000, 0x63000000, 0xC6000000,
           0x97000000, 0x35000000, 0x6A000000, 0xD4000000,
           0xB3000000, 0x7D000000, 0xFA000000, 0xEF000000,
           0xC5000000, 0x91000000, 0x39000000, 0x72000000,
           0xE4000000, 0xD3000000, 0xBD000000, 0x61000000,
           0xC2000000, 0x9F000000, 0x25000000, 0x4A000000,
           0x94000000, 0x33000000, 0x66000000, 0xCC000000,
           0x83000000, 0x1D000000, 0x3A000000, 0x74000000,
           0xE8000000, 0xCB000000, 0x8D000000}
```

```
Sbox = {
    { 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76 } ,
    { 0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0 } ,
    { 0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15 } ,
    { 0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75 } ,
    { 0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84 } ,
    { 0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf } ,
    { 0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8 } ,
    { 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2 } ,
    { 0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73 } ,
    { 0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb } ,
    { 0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79 } ,
    { 0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08 } ,
    { 0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a } ,
    { 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e } ,
    { 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf } ,
    { 0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 }
    };
```

```
InvSbox = {
    { 0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb } ,
    { 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb } ,
    { 0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e } ,
    { 0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25 } ,
    { 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92 } ,
    { 0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84 } ,
    { 0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06 } ,
    { 0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b } ,
    { 0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73 } ,
    { 0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e } ,
    { 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b } ,
    { 0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4 } ,
    { 0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f } ,
    { 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef } ,
    { 0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61 } ,
    { 0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d }
    };
```

Unit Tests

I strongly encourage you to use unit tests as you develop your code. Here are some test cases.

Finite Field Arithmetic

```
ffAdd(0x57,0x83) == 0xd4


xtime(0x57) == 0xae
xtime(0xae) == 0x47
xtime(0x47) == 0x8e
xtime(0x8e) == 0x07

ffMultiply(0x57,0x13) == 0xfe
```

Key Expansion

The first set of unit tests cover the methods used by key expansion -- `subWord` and `rotWord`. The second part tests the expansion of a 128-bit cipher key as shown in A.1.

```
subWord(0x00102030) == 0x63cab704
subWord(0x40506070) == 0x0953d051
subWord(0x8090a0b0) == 0xcd60e0e7
subWord(0xc0d0e0f0) == 0xba70e18c

rotWord(0x09cf4f3c) == 0xcf4f3c09
rotWord(0x2a6c7605) == 0x6c76052a

uint8_t key[16] =       { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
                          0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c };
uint32_t expanded[44] = { 0x2b7e1516, 0x28aed2a6, 0xabf71588, 0x09cf4f3c,
                          0xa0fafe17, 0x88542cb1, 0x23a33939, 0x2a6c7605,
                          0xf2c295f2, 0x7a96b943, 0x5935807a, 0x7359f67f,
                          0x3d80477d, 0x4716fe3e, 0x1e237e44, 0x6d7a883b,
                          0xef44a541, 0xa8525b7f, 0xb671253b, 0xdb0bad00,
                          0xd4d1c6f8, 0x7c839d87, 0xcaf2b8bc, 0x11f915bc,
                          0x6d88a37a, 0x110b3efd, 0xdbf98641, 0xca0093fd,
                          0x4e54f70e, 0x5f5fc9f3, 0x84a64fb2, 0x4ea6dc4f,
                          0xead27321, 0xb58dbad2, 0x312bf560, 0x7f8d292f,
                          0xac7766f3, 0x19fadc21, 0x28d12941, 0x575c006e,
                          0xd014f9a8, 0xc9ee2589, 0xe13f0cc8, 0xb6630ca6 };

uint32_t w[44];
keyExpansion(key,w);
/* Test that w == expanded */
```

Cipher

This tests the cipher example shown in Appendix B. The first tests are for individual methods (subBytes, shiftRows, mixColumns, addRoundKey) and for only round 1, the first full round. The last test is for the entire cipher example.

```
uint8_t state[4][4] =  { {0x19,0xa0,0x9a,0xe9},
                         {0x3d,0xf4,0xc6,0xf8},
                         {0xe3,0xe2,0x8d,0x48},
                         {0xbe,0x2b,0x2a,0x08}};

uint8_t sub[4][4] =    { {0xd4,0xe0,0xb8,0x1e},
                         {0x27,0xbf,0xb4,0x41},
                         {0x11,0x98,0x5d,0x52},
                         {0xae,0xf1,0xe5,0x30}};

uint8_t shift[4][4] =  { {0xd4, 0xe0, 0xb8, 0x1e},
                         {0xbf, 0xb4, 0x41, 0x27},
                         {0x5d, 0x52, 0x11, 0x98},
                         {0x30, 0xae, 0xf1, 0xe5}};

uint8_t mix[4][4] =    { {0x04, 0xe0, 0x48, 0x28},
                         {0x66, 0xcb, 0xf8, 0x06},
                         {0x81, 0x19, 0xd3, 0x26},
                         {0xe5, 0x9a, 0x7a, 0x4c}};

uint8_t round[4][4] =   { {0xa4, 0x68, 0x6b, 0x02},
                          {0x9c, 0x9f, 0x5b, 0x6a},
                          {0x7f, 0x35, 0xea, 0x50},
                          {0xf2, 0x2b, 0x43, 0x49}};

subBytes(state);
/* Test that state == sub */
shiftRows(state);
/* Test that state == shift */
mixColumns(state);
/* Test that state == mix */
addRoundKey(state,w,4);
/* Test that state == round */

uint8_t in[16]  = { 0x32, 0x43, 0xf6, 0xa8, 0x88, 0x5a, 0x30, 0x8d,
                    0x31, 0x31, 0x98, 0xa2, 0xe0, 0x37, 0x07, 0x34 };
uint8_t out[16] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t result[16] = { 0x39, 0x25, 0x84, 0x1d, 0x02, 0xdc, 0x09, 0xfb,
                       0xdc, 0x11, 0x85, 0x97, 0x19, 0x6a, 0x0b, 0x32 };
cipher(in,out,w);
/* Test that out == result */
```