Assignment 5

Ethan Fidler 2/15/2023

Output

```
Microsoft Windows [Version 10.0.19044.2486]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Ethan Fidler\Desktop\Data Encoding\CS5125> cmd /C ""C:\Users\Ethan Fidler\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.5.8-hotspot\bin\java.exe" -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\Ethan Fidler\AppData\Roaming\Code\User\workspaceStorage\fe8f402d851bca048f3125949912f681\redhat.java\jdt_ws\CS5125_68b77586\bin" DE8A "
message
4i(├G₧y╟Σ3╢!½▼<
```

```
C:\Users\Ethan Fidler\Desktop\Data Encoding\CS5125> cmd /C ""C:\Users\Ethan Fidler\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.5.8-hotspot\bin\java.exe" -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\Ethan Fidler\AppData\Roaming\Code\User\workspaceStorage\fe8f402d851bca048f3125949912f681\redhat.java\jdt_ws\CS5125_68b77586\bin" DE8B "
4i(├G₧y╟Σ3╢!½▼<
message
p»≤w├░FSß━▲sgb±α
```

I don't think The test files were properly or consistently readable by my computer or IDE and I am not confident in me having correctly used them.

```
C:\Users\Ethan Fidler\Desktop\Data Encoding\CS5125> cmd /C ""C:\Users\Ethan Fidler\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.5.8-hotspot\bin\java.exe" -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\Ethan Fidler\AppData\Roaming\Code\User\workspaceStorage\fe8f402d851bca048f3125949912f681\redhat.java\jdt_ws\CS5125_68b77586\bin" DE8B "
bb6b c679 4e2c 74ce 848e 91b5 3e9a 7bd5
╘.BÄ0[ñì~cªB↑îê|Sí⌠ç;√xU²3⌠c↓»íf13d 40b9 5c82 7187 7f91 4f84 b2fa d9f2
φ┴Xj9,┬╤/ò<ÿ¤3`º+≤D@Kâ╖▼☻■╒?D D²÷*w.z.f«}à╢æp»≤w├░FSß━▲sgb±αec01 6fd5 b627 28dd d13f 190d a45e 6577
m≡▼\5æ-πîn(ú╓iHB→_»dü½I≈°ïsß├ΩSæ¤X      m→!g¬.╟╫║╟6921 74f7 aa93 428b d743 6840 c6ed 12a0
│ª┴╧<K▓,Θ&┤♀l¥æ▲╬\ô\ák·v(╘d╧òTF▌5e8a d403 099a 6210 ad28 37a9 bac5 e375
♫-53«⌡σ<⌐ï«B-√₧┘Φmx▐ò#N⌡çT↕♦ë░╤↕·DÆ▓v«°↔}▓¶╢ÑÆú↑p»≤w├░FSß━▲sgb±α7774 3571 40b1 1553 f734 591f c6c1 03b7
╤♂¢2↕C¤≈Ä╤lhv4 `Ç⌠Br┘î▄τ►5Θ♠╘≤╤=9G1╪ì♀▲°#╢=τáhI¶baa0 f503 e3ce ab93 c57a dd68 fd11 5785
áá|V5░╓Zi1tÖ▀¢(ö1║â╓[VⁿjF~↕ä    ≥úα
ç'►┴²É▼↕ÉS{╚93cd 5356 b04c e7f9 a8f1 3fca cf5e 3278
ú4@     ^X)êK━≈fóréiⁿ♂G╕¶ⁿb◄¥¢₧>ª└►≡·Γ┘½¥W▲─?á■╪?6╕↔68ef f041 b455 5915 0bdb 9421 def0 2f1a
óπK╣dπNâ░Σx♫░♂!u#½┤í¥▐¶Ncp▼ΦS↕x@5b30 9869 4cc0 b82c ee8a d3b7 8bda 0acb
,╟BCΘ=αTÄZ≥Æ¬·X╢↓►♀▐┴²Z♀#t1
,.╘½u├lφ4Σ»~╨║v│¿│▀╡8ccb e3b3 ead2 dc81 154b d193 7a69 54e1
╚Θç░Ñ₧,╤π╕ⁿ╠╪5♠╒↓╧e0í↓¶pê╤α[#/ät817e bab3 f5c4 76ef 294d 01b5 c259 da2d
MéUÜRp3zêò!εùWë2Tn╧ƒΩùz▐►äåEs]$tÄ k‼≤T┘ïτ¢£☻┘û±↓3b98 4a58 9297 52a7 d70e 17d8 d71d 4917
fï\{Næ♣┤º=↕↔╚♫┘1‼εIòSU↕▐I`9╚#α╓τ╘⌡↔ ─⌐I¬─║┐w│━╤£7e00 7a93 2093 f698 3f70 ab7f 2566 9b5e
```

Code

DE8A

```
import java.io.*;
import java.util.*;

public class DE8A{

  static final int numberOfBits = 8;
  static final int fieldSize = 1 << numberOfBits;
  static final int irreducible = 0x11b;
  static final int logBase = 3;
  static final byte[][] A = new byte[][] {
            {1, 1, 1, 1, 1, 0, 0, 0},
            {0, 1, 1, 1, 1, 1, 0, 0},
        {0, 0, 1, 1, 1, 1, 1, 0},
            {0, 0, 0, 1, 1, 1, 1, 1},
        {1, 0, 0, 0, 1, 1, 1, 1},
```

```
            {1, 1, 0, 0, 0, 1, 1, 1},
            {1, 1, 1, 0, 0, 0, 1, 1},
            {1, 1, 1, 1, 0, 0, 0, 1}
    };
  static final byte[] B = new byte[] { 0, 1, 1, 0, 0, 0, 1, 1};
  static final byte[][] G = new byte[][] {
            {2, 1, 1, 3},
            {3, 2, 1, 1},
            {1, 3, 2, 1},
            {1, 1, 3, 2}
        };
  static String hexkey = "0f1571c947d9e8590cb7add6af7f6798";  // Stallings key in
"An AES Example"
  int[] alog = new int[fieldSize];
  int[] log = new int[fieldSize];
  int[] S = new int[fieldSize];
  static final int blockSize = 16;
  static final int numberOfRounds = 11;
  int[] state = new int[blockSize];
  int[][] roundKey = new int[numberOfRounds][blockSize];

  int modMultiply(int a, int b, int m){
    int product = 0;
    for (; b > 0; b >>= 1){
      if ((b & 1) > 0) product ^= a;
      a <<= 1;
      if ((a & fieldSize) > 0) a ^= m;
    }
    return product;
  }

  void makeLog(){
    alog[0] = 1;
    for (int i = 1; i < fieldSize; i++)
      alog[i] = modMultiply(logBase, alog[i - 1], irreducible);
    for (int i = 1; i < fieldSize; i++) log[alog[i]] = i;
  }

  int logMultiply(int a, int b){
    return (a == 0 || b == 0) ? 0 : alog[(log[a] + log[b]) % (fieldSize - 1)];
  }

  int multiplicativeInverse(int a){
    return alog[fieldSize - 1 - log[a]];
  }

  void buildS(){
     int[] bitColumn = new int[8];
     for (int i = 0; i < fieldSize; i++){
       int inverse = i < 2 ? i : multiplicativeInverse(i);
       for (int k = 0; k < 8; k++)
           bitColumn[k] = inverse >> (7 - k) & 1;
       S[i] = 0;
       for (int k = 0; k < 8; k++){
```

```
          int bit = B[k];
          for (int l = 0; l < 8; l++)
            if (bitColumn[l] == 1) bit ^= A[k][l];
          S[i] ^= bit << 7 - k;
        }
      }
    }

  int readBlock(){
    byte[] data = new byte[blockSize];
    int len = 0;
    try {
      len = System.in.read(data);
    } catch (IOException e){
      System.err.println(e.getMessage());
      System.exit(1);
    }
    if (len <= 0) return len;
    for (int i = 0; i < len; i++){
      if (data[i] < 0) state[i] = data[i] + fieldSize;
      else state[i] = data[i];
    }
    for (int i = len; i < blockSize; i++) state[i] = 0;
    return len;
  }

   void subBytes(){
     for (int i = 0; i < blockSize; i++)
       state[i] = S[state[i]];
   }

  void shiftRows(){
    int temp = state[2]; state[2] = state[10]; state[10] = temp;
    temp = state[6]; state[6] = state[14]; state[14] = temp;
    temp = state[1]; state[1] = state[5]; state[5] = state[9];
    state[9] = state[13]; state[13] = temp;
    temp = state[3]; state[3] = state[15]; state[15] = state[11];
    state[11] = state[7]; state[7] = temp;
  }

   void mixColumns(){
    int[] temp = new int[4];
    for (int k = 0; k < 4; k++){
     for (int i = 0; i < 4; i++){
       temp[i] = 0;
       for (int j = 0; j < 4; j++)
         temp[i] ^= logMultiply(G[j][i], state[k * 4 + j]);
     }
     for (int i = 0; i < 4; i++) state[k * 4 + i] = temp[i];
    }
   }

  void expandKey(){
    for (int i = 0; i < blockSize; i++) roundKey[0][i] =
```

```
      Integer.parseInt(hexkey.substring(i * 2, (i + 1) * 2), 16);
    int rcon = 1;
    for (int i = 1; i < numberOfRounds; i++){
      roundKey[i][0] = S[roundKey[i-1][13]] ^ rcon;
      rcon <<= 1; if (rcon > 0xFF) rcon ^= irreducible;
      roundKey[i][1] = S[roundKey[i-1][14]];
      roundKey[i][2] = S[roundKey[i-1][15]];
      roundKey[i][3] = S[roundKey[i-1][12]];
      for (int k = 0; k < 4; k++)
         roundKey[i][k] ^= roundKey[i-1][k];
      for (int k = 4; k < blockSize; k++)
         roundKey[i][k] = roundKey[i][k-4] ^ roundKey[i-1][k];
    }
  }

  void addRoundKey(int round){
    for (int k = 0; k < blockSize; k++)
       state[k] ^= roundKey[round][k];
  }

   void blockCipher(){
     addRoundKey(0);
     for (int i = 1; i < numberOfRounds; i++){
       subBytes();
       shiftRows();
       if (i < numberOfRounds - 1) mixColumns();
       addRoundKey(i);
     }
   }

  void writeBlock(){
    byte[] data = new byte[blockSize];
    for (int i = 0; i < blockSize; i++)
      data[i] = (byte)(state[i]);
    try {
      System.out.write(data);
    } catch (IOException e){
      System.err.println(e.getMessage());
      System.exit(1);
    }
  }

   void encrypt(){
     while (readBlock() > 0){
       blockCipher();
       writeBlock();
     }
     System.out.flush();
   }


public static void main(String[] args){
   DE8A de8 = new DE8A();
   de8.makeLog();
```

```
    de8.buildS();
    de8.expandKey();
    de8.encrypt();
}
}
```

DE8B

```
// DE8B.java CS5125/6025 Cheng 2023
// Implementing AES decryption
// Usage: java DE8B < encrypted > original

import java.io.*;
import java.util.*;

public class DE8B{

  static final int numberOfBits = 8;
  static final int fieldSize = 1 << numberOfBits;
  static final int irreducible = 0x11b;
  static final int logBase = 3;
  static final byte[][] A = new byte[][] {
            {1, 1, 1, 1, 1, 0, 0, 0},
            {0, 1, 1, 1, 1, 1, 0, 0},
        {0, 0, 1, 1, 1, 1, 1, 0},
            {0, 0, 0, 1, 1, 1, 1, 1},
        {1, 0, 0, 0, 1, 1, 1, 1},
            {1, 1, 0, 0, 0, 1, 1, 1},
            {1, 1, 1, 0, 0, 0, 1, 1},
            {1, 1, 1, 1, 0, 0, 0, 1}
    };
  static final byte[] B = new byte[] { 0, 1, 1, 0, 0, 0, 1, 1};
  static final byte[][] Gi = new byte[][] {
            {14, 9, 13, 11},
            {11, 14, 9, 13},
            {13, 11, 14, 9},
            {9, 13, 11, 14}
        };
  static String hexkey = "0f1571c947d9e8590cb7add6af7f6798";  // Stallings key in
"An AES Example"
  int[] alog = new int[fieldSize];
  int[] log = new int[fieldSize];
  int[] S = new int[fieldSize];
  int[] Si = new int[fieldSize];
  static final int blockSize = 16;
  static final int numberOfRounds = 11;
  int[] state = new int[blockSize];
  int[][] roundKey = new int[numberOfRounds][blockSize];

  int modMultiply(int a, int b, int m){
    int product = 0;
```

```
    for (; b > 0; b >>= 1){
      if ((b & 1) > 0) product ^= a;
      a <<= 1;
      if ((a & fieldSize) > 0) a ^= m;
    }
    return product;
  }

  void makeLog(){
    alog[0] = 1;
    for (int i = 1; i < fieldSize; i++)
      alog[i] = modMultiply(logBase, alog[i - 1], irreducible);
    for (int i = 1; i < fieldSize; i++) log[alog[i]] = i;
  }

  int logMultiply(int a, int b){
    return (a == 0 || b == 0) ? 0 : alog[(log[a] + log[b]) % (fieldSize - 1)];
  }

  int multiplicativeInverse(int a){
    return alog[fieldSize - 1 - log[a]];
  }

  void buildS(){
     int[] bitColumn = new int[8];
     for (int i = 0; i < fieldSize; i++){
       int inverse = i < 2 ? i : multiplicativeInverse(i);
       for (int k = 0; k < 8; k++)
           bitColumn[k] = inverse >> (7 - k) & 1;
       S[i] = 0;
       for (int k = 0; k < 8; k++){
          int bit = B[k];
          for (int l = 0; l < 8; l++)
            if (bitColumn[l] == 1) bit ^= A[k][l];
          S[i] ^= bit << 7 - k;
       }
       Si[S[i]] = i;
    }
  }

 int readBlock(){
   byte[] data = new byte[blockSize];
   int len = 0;
   try {
     len = System.in.read(data);
   } catch (IOException e){
     System.err.println(e.getMessage());
     System.exit(1);
   }
   if (len <= 0) return len;
   for (int i = 0; i < len; i++){
     if (data[i] < 0) state[i] = data[i] + fieldSize;
     else state[i] = data[i];
   }
```

```
    for (int i = len; i < blockSize; i++) state[i] = 0;
    return len;
  }

   void inverseSubBytes(){
     for (int i = 0; i < blockSize; i++)
       state[i] = Si[state[i]];
   }

  void inverseShiftRows(){
     // [0, 4, 8, 12]
     // [1, 5, 9, 13] -> [5, 9, 13, 1]
     // [2, 6, 10, 14] -> [10, 14, 2, 6]
     // [3, 7, 11, 15] -> [15, 3, 7, 11]
     int temp = state[13]; state[13] = state[9]; state[9] = state[5]; state[5] =
state[1]; state[1] = temp;
     temp = state[10]; state[10] = state[2]; state[2] = temp;
     temp = state[14]; state[14] = state[6]; state[6] = temp;
     temp = state[7]; state[7] = state[11]; state[11] = state[15]; state[15] =
state[3]; state[3] = temp;

  }

   void inverseMixColumns(){
    int[] temp = new int[4];
    for (int k = 0; k < 4; k++){
     for (int i = 0; i < 4; i++){
       temp[i] = 0;
       for (int j = 0; j < 4; j++)
         temp[i] ^= logMultiply(Gi[j][i], state[k * 4 + j]);
     }
     for (int i = 0; i < 4; i++) state[k * 4 + i] = temp[i];
    }
   }

  void expandKey(){
    for (int i = 0; i < blockSize; i++) roundKey[0][i] =
      Integer.parseInt(hexkey.substring(i * 2, (i + 1) * 2), 16);
    int rcon = 1;
    for (int i = 1; i < numberOfRounds; i++){
      roundKey[i][0] = S[roundKey[i-1][13]] ^ rcon;
      rcon <<= 1; if (rcon > 0xFF) rcon ^= irreducible;
      roundKey[i][1] = S[roundKey[i-1][14]];
      roundKey[i][2] = S[roundKey[i-1][15]];
      roundKey[i][3] = S[roundKey[i-1][12]];
      for (int k = 0; k < 4; k++)
         roundKey[i][k] ^= roundKey[i-1][k];
      for (int k = 4; k < blockSize; k++)
         roundKey[i][k] = roundKey[i][k-4] ^ roundKey[i-1][k];
    }
  }

  void inverseAddRoundKey(int round){
    for (int k = 0; k < blockSize; k++)
```

```
      state[k] ^= roundKey[numberOfRounds-1-round][k];   // you need to figure out
what "?" is
  // round 0 in decoder uses roundKey[numberOfRounds - 1]
  // round 1 in decoder uses roundKey[numberOfRounds - 2]
  // ... round 10 uses roundKey[0]
 }

  void blockDecipher(){
    inverseAddRoundKey(0);
    for (int i = 1; i < numberOfRounds; i++){
      inverseSubBytes();
      inverseShiftRows();
      inverseAddRoundKey(i);
      if (i < numberOfRounds - 1) inverseMixColumns();
    }
  }

 void writeBlock(){
   byte[] data = new byte[blockSize];
   for (int i = 0; i < blockSize; i++)
     data[i] = (byte)(state[i]);
   try {
     System.out.write(data);
   } catch (IOException e){
     System.err.println(e.getMessage());
     System.exit(1);
   }
 }

 void decrypt(){
   while (readBlock() > 0){
     blockDecipher();
     writeBlock();
   }
   System.out.flush();
 }


public static void main(String[] args){
   DE8B de8 = new DE8B();
   de8.makeLog();
   de8.buildS();
   de8.expandKey();
   de8.decrypt();
}
}
```