MSCS 630 – SECURITY ALGORITHMS AND PROTOCOLS

Lab 5

APRIL 18, 2018

THOMPSON RAJAN
CWID: 20082947

Generating AES Cipher Text

1. AESCipher.java

```java
/**
 * File: AESCipher.java
 * Author: Thompson Rajan
 * Course: MSCS 630 Security Algorithms and Protocols
 * Assignment: Lab 5
 * Due Date: Wednesday, April 18, 2018
 * Version: 1.0
 *
 * This file contains the AES Cipher.
 */

/**
 * This class contains the static methods for the AES Cipher.
 */
public class AESCipher {

  /**
   * This static method generates the 11 round keys for the AES cipher.
   * @param keyHex - Input Key
   * @return - String array of 11 round keys
   */
  static String[] aesRoundKeys(String keyHex){
    // Array to store the 11 round keys.
    String[] roundKeysHex = new String[11];

    // Initialize string array.
    for(int i = 0; i < roundKeysHex.length;i++){
      roundKeysHex[i] = "";
    }

    // This array is used extensively to operate on each round key values.
    int[][] roundKeys = new int[4][44];
    int k = 0;

    // Get key value by individual bytes and store in the first four columns.
    for(int i = 0; i < 4 ; i++){
      for(int j = 0; j < 4; j++){
          roundKeys[j][i] = Integer.parseInt(keyHex.substring(k, k+2)
                  ,16);
          k += 2;
      }
    }

    // Vectors operated with integer decimal values.
    for(int i = 4; i < 44; i++ ){

      // XOR last 4th column and the previous column and then store the
      // result in the current column leaving the first column of the round key.
      if (i % 4 != 0) {
        for (int j = 0; j < 4; j++) {
          roundKeys[j][i] = roundKeys[j][i - 4] ^ roundKeys[j][i - 1];
        }
      }
      else {
        int round = i / 4;
        int rCon = Integer.parseInt(aesRCon(round), 16);

        // Temporary vector to evaluate the new vector.
        int[] tempKey = new int[4];

        // Left Shift by 1 and S-Box transformation.
        for(int j = 0; j < 4; j++){
          tempKey[(j + 3) % 4] = Integer.parseInt(aesSBox(
                  Integer.toHexString(roundKeys[j][i - 1])), 16);
        }

        // XOR first element with corresponding Rijndael constant from the
        // rCon table
        tempKey[0] = tempKey[0] ^ rCon;

        // XOR new vector with the last 4th column of the key vector.
        for (int j = 0; j < 4; j++) {
          roundKeys[j][i] = roundKeys[j][i - 4] ^ tempKey[j];
        }
      }
    }

    // Implementation to convert results to hexadecimal string array.
    for(int i = 0; i < 44; i++){
      int round = i / 4;
      for(int j = 0; j < 4; j++){

        // Padding 0 before single digit strings wherever possible.
        if(Integer.toHexString(roundKeys[j][i]).length() == 1){
          if(Integer.toHexString(roundKeys[j][i]).equals("0")){
            roundKeysHex[round] += "00";
          }
          else
            roundKeysHex[round]+= "0" + Integer.toHexString(roundKeys[j][i])
                    .toUpperCase();
        }
        else {
          roundKeysHex[round] += Integer.toHexString(roundKeys[j][i])
                  .toUpperCase();
        }
      }
    }
    return roundKeysHex;
  }

  /**
   * This method transforms does the S-Box transform based on the sBox table.
   * @param inHex - Input hex string value
   * @return - Transformed hex string value
   */
  static String aesSBox(String inHex) {

    int it = Integer.parseInt(inHex,16);
    char x = s[it];

    return Integer.toHexString((int) x).toUpperCase();
  }
```

```
/**
 * This method gets the Rijndael key based on the round number.
 * @param round - Current round number
 * @return - Rijndael key hex string value.
 */
static String aesRCon(int round){
  char x = r[round];

  return Integer.toHexString((int) x).toUpperCase();
}

/**
 * This method converts a unicode hex string to a 4 x 4 hex array.
 * @param unicode32ByteString - 32 Byte Unicode String
 * @return - 4 x 4 Hex Array
 */
static char[][] covertTo4x4HexArray(String unicode32ByteString){

  char out16ByteHexArray[][] = new char[4][4];

  int k = 0;
  for(int i = 0; i < 4 ; i++){
    for(int j = 0; j < 4; j++){
        out16ByteHexArray[j][i] = (char) Integer.parseInt(unicode32ByteString
                        .substring(k, k+2)
                ,16);
        k += 2;
    }
  }
  return out16ByteHexArray;
}

/**
 * This method converts a 4 x 4 hex array to a 32 byte unicode string.
 * @param hex4x4Array - 4x4 Hex Array
 * @return - 32 Byte Unicode String
 */
static String convert2String(char[][] hex4x4Array){
  String hex32ByteString = "";

  for(int i = 0; i < 4; i++){
    for(int j = 0; j < 4; j++){
      if(Integer.toHexString(hex4x4Array[j][i]).toUpperCase().length() == 1){
        hex32ByteString += "0";
      }
      hex32ByteString += Integer.toHexString(hex4x4Array[j][i]).toUpperCase();
    }
  }
  return hex32ByteString;
}

/**
 * This matrix performs the Add Key operation by XORing the input matrices.
 * @param sHex - Partially processed value
 * @param keyHex - Round Key
 * @return - XORed matrix
 */
static char[][] AESStateXOR(char sHex[][], char keyHex[][]){

  char outStateHex[][] = new char[4][4];
```

```
    for(int i = 0; i < 4; i++){
      for(int j = 0; j < 4; j++){
        outStateHex[i][j] = (char) (sHex[i][j] ^ keyHex[i][j]);
      }
    }
    return outStateHex;
  }

  /**
   * This method translates the input matrix values to S-Boxed values.
   * @param inStateHex - Input matrix
   * @return - S-Boxed matrix
   */
  static char[][] AESNibbleSub(char[][] inStateHex){

    char outHex[][] = new char[4][4];

    for(int i = 0; i < 4; i++){
      for(int j = 0; j < 4; j++){
        outHex[i][j] = (char)
                Integer.parseInt(aesSBox(Integer.toHexString(
                        inStateHex[i][j])),16);
      }
    }
    return outHex;
  }

  /**
   * This method performs the increasing row-wise shift left steps down the
   * row on a matrix.
   * @param inStateHex - Input matrix
   * @return - Shifted matrix
   */
  static char[][] AESShiftRow(char[][] inStateHex){

    char[][] outStateHex = inStateHex;

    char temp = 0;
    for(int i = 0; i < 4; i++){
      // Shift left steps increases with rows
      for(int k = 0; k < i; k++) {
        for (int j = 0; j < 4; j++) {
          if (j == 0) {
            temp = inStateHex[i][j];
          }
          if ((j + 1) < 4) {
            inStateHex[i][j] = inStateHex[i][j + 1];
          }
          if (j == 3) {
            inStateHex[i][j] = temp;
          }
        }
      }
    }
    return outStateHex;
  }

  /**
   * This method performs the Rijdael MixColumns operation, byte multiplying
```

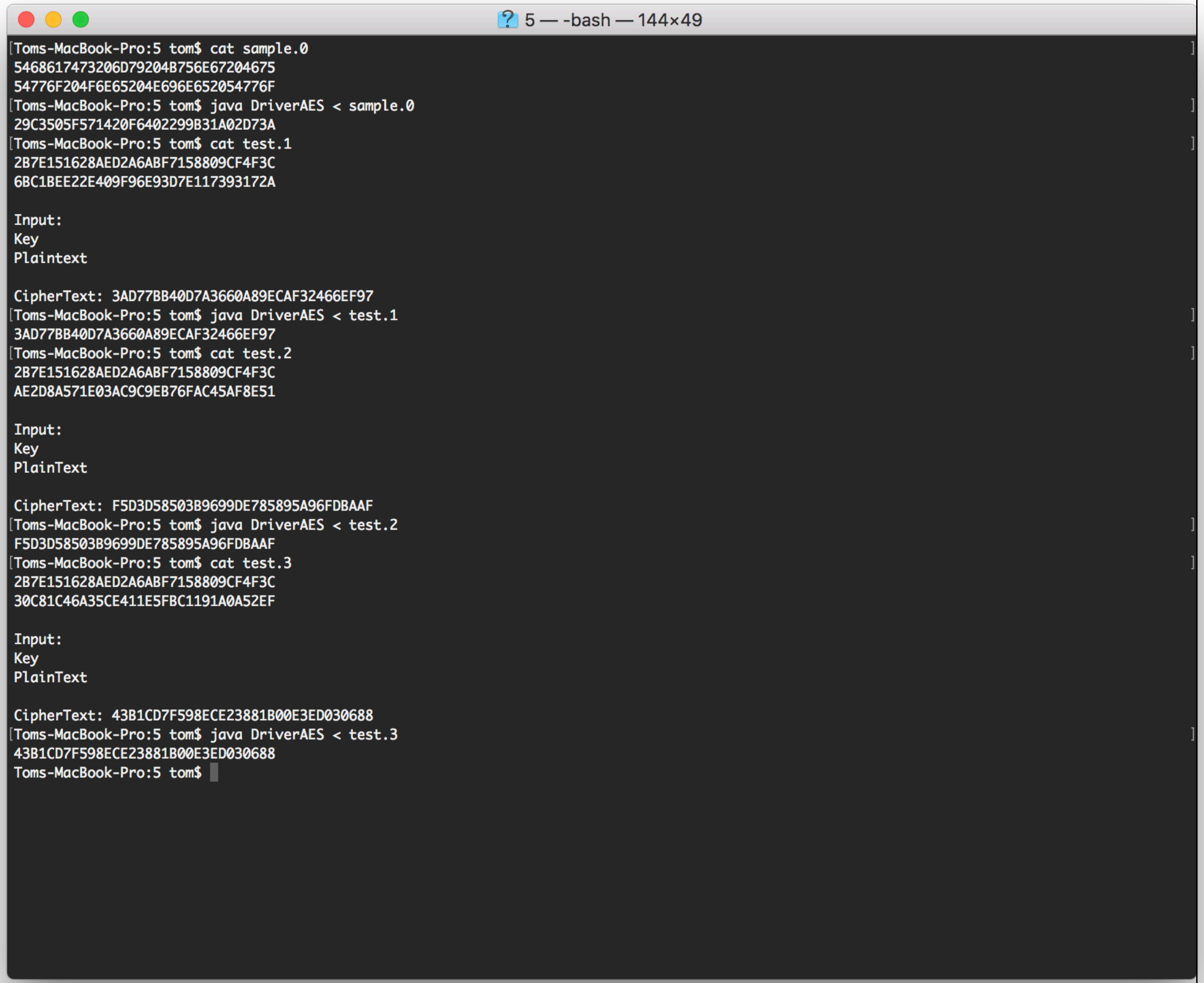
```
 * with Galois multiples.
 * @param inStateHex - Partially processed value (probably after row shifts)
 * @return - Matrix mixed by column.
 */
static char[][] AESMixColumn(char[][] inStateHex){

  char[][] outStateHex =  inStateHex;

  char a[] = new char[4];
  char r[] = new char[4];

  for(int i = 0; i < 4; i++){
    for(int j = 0; j < 4; j++){
      a[j] = inStateHex[j][i];
    }
    // Byte matrix multiplication
    r[0] = (char)(gMult2[a[0]] ^ gMult3[a[1]] ^ a[2] ^ a[3]);
    r[1] = (char)(a[0] ^ gMult2[a[1]] ^ gMult3[a[2]] ^ a[3]);
    r[2] = (char)(a[0] ^ a[1] ^ gMult2[a[2]] ^ gMult3[a[3]]);
    r[3] = (char)(gMult3[a[0]] ^ a[1] ^ a[2] ^ gMult2[a[3]]);

    for(int j = 0; j < 4; j++) {
      outStateHex[j][i] = r[j];
    }
  }
  return outStateHex;
}

/**
 * This method performs the 10 round AES encryption cycles.
 * @param pTextHex - Plain Text
 * @param keyHex - Key
 * @return - AES Cipher Text
 */
static String AES(String pTextHex, String keyHex){

  String cTextHex="";
  char pTextArray[][] = covertTo4x4HexArray(pTextHex);

  //Generate round keys
  String roundKey[] = aesRoundKeys(keyHex);

  char processArray[][] = new char[4][4];
  processArray = AESStateXOR(pTextArray,covertTo4x4HexArray(roundKey[0]));

  // Repeats steps for 10 rounds
  for(int i = 0; i < 10; i++){

    processArray = AESNibbleSub(processArray);
    processArray = AESShiftRow(processArray);

    // Mix Columns just for 9 rounds
    if(i < 9) {
      processArray = AESMixColumn(processArray);
    }
    processArray = AESStateXOR(processArray, covertTo4x4HexArray(roundKey[i + 1]));
  }
  cTextHex = convert2String(processArray);

  return cTextHex;
```

```
    }

    // Galois Multiples by 2
    private static final char[] gMult2 = {
      0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,
      0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,
      0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,
      0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,
      0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,
      0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,
      0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,
      0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,
      0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,
      0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,
      0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,
      0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,
      0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,
      0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,
      0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,
      0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,
      0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,
      0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,
      0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,
      0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,
      0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,
      0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,
      0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,
      0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,
      0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,
      0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,
      0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,
      0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,
      0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,
      0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,
      0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,
      0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5
    };

    // Galois Multiples by 3
    private static final char[] gMult3 = {
      0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,
      0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,
      0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,
      0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,
      0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,
      0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,
      0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,
      0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,
      0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,
      0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,
      0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,
      0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,
      0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,
      0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,
      0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,
      0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,
      0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,
      0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,
      0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,
      0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,
      0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,
    0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xe9,0xea,
    0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,
    0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,
    0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,
    0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,
    0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,
    0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,
    0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,
    0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,
    0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,
    0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a
  };

  // S-Box Transformation Table
  private static final char[] s = {
    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
  };

  // Rijndael Key Schedule
  private static final char[] r = {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
    0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
    0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,
    0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
    0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,
    0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
    0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08,
    0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
    0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,
    0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
    0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61,
    0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
    0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,
    0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
    0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e,
    0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
    0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4,
    0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
    0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8,
    0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
    0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,
    0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
    0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91,
    0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
    0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d,
    0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
    0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c,
    0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
    0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa,
    0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
    0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66,
    0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d
  };
}
```

2. DriverAES.java

```java
/**
 * File: DriverAES.java
 * Author: Thompson Rajan
 * Course: MSCS 630 Security Algorithms and Protocols
 * Assignment: Lab 5
 * Due Date: Wednesday, April 18, 2018
 * Version: 1.0
 *
 * This file contains a class that calls the AESCipher to generate the
 * ciphertext.
 */

import java.util.Scanner;

/**
 * This class calls the AESCipher to generate the ciphertext.
 */
public class DriverAES {
  /**
   * This method makes a static call to AES to get the ciphertext for a given
   * plaintext and a key.
   * @param args - null
   */
  public static void main(String[] args) {

    Scanner in = new Scanner(System.in);

    String keyHex = in.nextLine();
    String pTextHex = in.nextLine();

    // Get Ciphertext
    String cTextHex = AESCipher.AES(pTextHex,keyHex);

    System.out.println(cTextHex);
  }
}
```

Output:



```
[Toms-MacBook-Pro:5 tom$ cat sample.0
5468617473206D79204B756E67204675
54776F204F6E65204E696E652054776F
[Toms-MacBook-Pro:5 tom$ java DriverAES < sample.0
29C3505F571420F6402299B31A02D73A
[Toms-MacBook-Pro:5 tom$ cat test.1
2B7E151628AED2A6ABF7158809CF4F3C
6BC1BEE22E409F96E93D7E117393172A

Input:
Key
Plaintext

CipherText: 3AD77BB40D7A3660A89ECAF32466EF97
[Toms-MacBook-Pro:5 tom$ java DriverAES < test.1
3AD77BB40D7A3660A89ECAF32466EF97
[Toms-MacBook-Pro:5 tom$ cat test.2
2B7E151628AED2A6ABF7158809CF4F3C
AE2D8A571E03AC9C9EB76FAC45AF8E51

Input:
Key
PlainText

CipherText: F5D3D58503B9699DE785895A96FDBAAF
[Toms-MacBook-Pro:5 tom$ java DriverAES < test.2
F5D3D58503B9699DE785895A96FDBAAF
[Toms-MacBook-Pro:5 tom$ cat test.3
2B7E151628AED2A6ABF7158809CF4F3C
30C81C46A35CE411E5FBC1191A0A52EF

Input:
Key
PlainText

CipherText: 43B1CD7F598ECE23881B00E3ED030688
[Toms-MacBook-Pro:5 tom$ java DriverAES < test.3
43B1CD7F598ECE23881B00E3ED030688
Toms-MacBook-Pro:5 tom$
```