FCC200 Report

Affine Cipher and S-DES Implementation

Connor Beardsmore - 15504319

Curtin University
Science and Engineering
Perth, Australia
April 2017

Affine Cipher

Compute Eligible Keys

There are two keys required, *a* and *b*. The first is required to be *coprime* with the length of the alphabet, in this scenario *26*. The second key representing the linear shift must be both positive and less than the length of the alphabet. To check if the *a* value is coprime, the following greatest common denominator check was utilized. If the greatest common denominator of *a* and 26 is 1, the value of *a* is *coprime* and the key is valid in combination with any valid *b* value.

```
//----------------------------------------------------------------------------
// FUNCTION: gcd
// IMPORT: a (int), b (int)
// PURPOSE: Find greatest common denominator of 2 numbers

int gcdFunction( int a, int b )
{
    int quotient, residue, temp, gcd = 1;
    // SWAP ELEMENTS TO GET THE MAX
    if ( a < b )
    {
        temp = a;
        a = b;
        b = temp;
    }
    // CHECK IF EITHER NUMBER IS 0
    if ( a == 0 )   return b;
    if ( b == 0 )   return a;
    // SATISFY THE EQUATION: A = B * quotient + residue
    quotient = a / b;
    residue = a - ( b * quotient );
    // RECURSIVELY CALL GCD
    gcd = gcdFunction( b, residue );
    return gcd;
}

//----------------------------------------------------------------------------
```

The results of calling this function on all *a* values from 1 to 25 are as follows:

- **gcdFunction(1, 26) = 1**
- gcdFunction(2, 26) = 2
- **gcdFunction(3, 26) = 1**
- gcdFunction(4, 26) = 2
- **gcdFunction(5, 26) = 1**
- gcdFunction(6, 26) = 2
- **gcdFunction(7, 26) = 1**
- gcdFunction(8, 26) = 2
- **gcdFunction(9, 26) = 1**
- gcdFunction(10, 26) = 2
- **gcdFunction(11, 26) = 1**
- gcdFunction(12, 26) = 2
- gcdFunction(13, 26) = 13
- gcdFunction(14, 26) = 2
- **gcdFunction(15, 26) = 1**
- gcdFunction(16, 26) = 2

- **gcdFunction(17, 26) = 1**
- gcdFunction(18, 26) = 2
- **gcdFunction(19, 26) = 1**
- gcdFunction(20, 26) = 2
- **gcdFunction(21, 26) = 1**
- gcdFunction(22, 26) = 2
- **gcdFunction(23, 26) = 1**
- gcdFunction(24, 26) = 2
- **gcdFunction(25, 26) = 1**

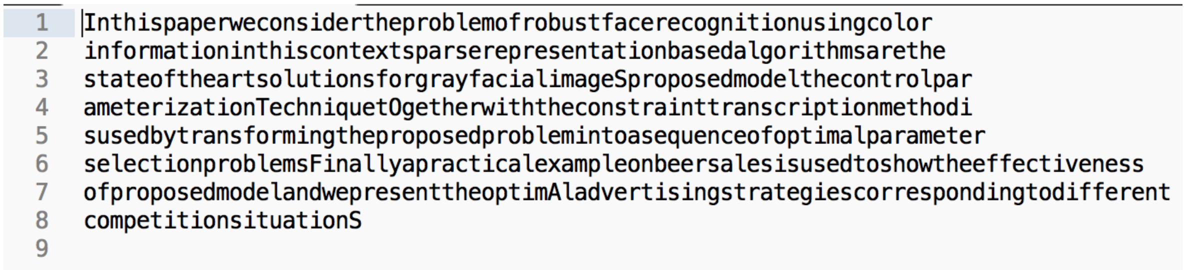
The full list of valid a values is: **1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25**

There are a total of 12 possible a values that are coprime with *26*. Each of these values can have a shift value (b) of 0 to 25. Thus, the total number of eligible keys is:

$$12 * 26 = 312$$

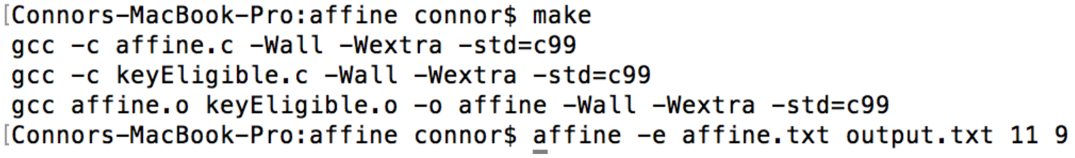
Of these, 26 keys are trivial Caesar ciphers and 286 are non-trivial.

Recovered Plaintext



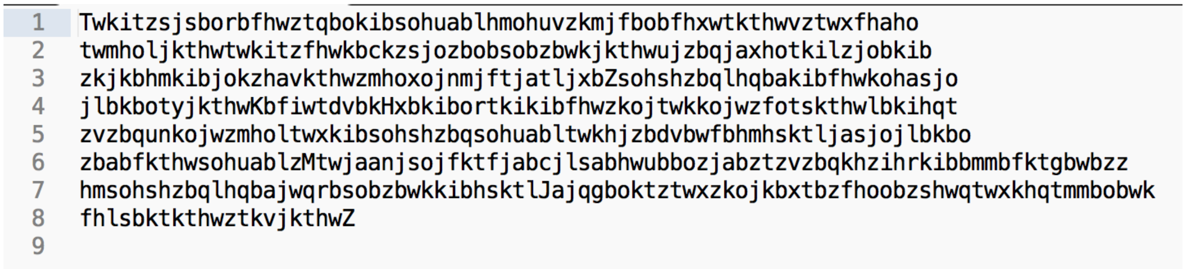
```
Inthispaperweconsidertheproblemofrobustfacerecognitionusingcolor
informationinthiscontextsparserepresentationbasedalgorithmsarethe
stateoftheartsolutionsforgrayfacialimageSproposedmodelthecontrolpar
ameterizationTechniquetOgetherwiththeconstrainttranscriptionmethodi
susedbytransformingtheproposedproblemintoasequenceofoptimalparameter
selectionproblemsFinallyapracticalexampleonbeersalesisusedtoshowtheeffectiveness
ofproposedmodelandwepresenttheoptimAladvertisingstrategiescorrespondingtodifferent
competitionsituationS
```

Figure 1: Original Plaintext File



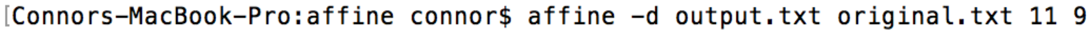
```
[Connors-MacBook-Pro:affine connor$ make
 gcc -c affine.c -Wall -Wextra -std=c99
 gcc -c keyEligible.c -Wall -Wextra -std=c99
 gcc affine.o keyEligible.o -o affine -Wall -Wextra -std=c99
[Connors-MacBook-Pro:affine connor$ affine -e affine.txt output.txt 11 9
```

Figure 2: Encryption Process



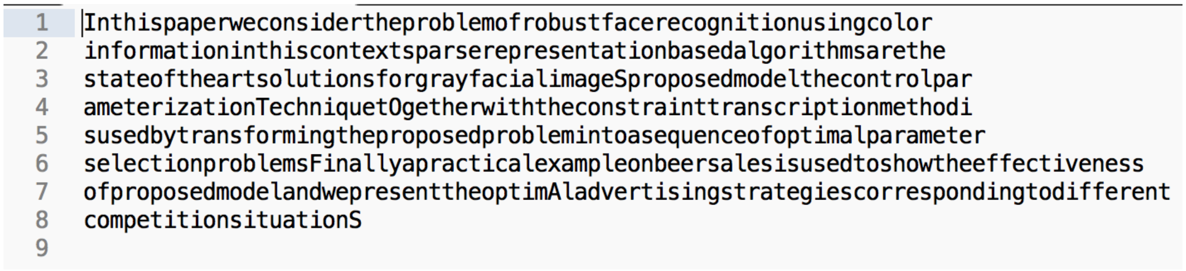
```
Twkitzsjsborbfhwztqbokibsohuablhmohuvzkmjfbobfhxwtkthwvztwxfhaho
twmholjkthwtwkitzfhwkbckzsjozbobsobzbwkjkthwujzbqjaxhotkilzjobkib
zkjkbhmkibjokzhavkthwzmhoxojnmjftjatljxbZsohshzbqlhqbakibfhwkohasjo
jlbkbotyjkthwKbfiwtdvbkHxbkibortkikibfhwzkojtwkkojwzfotskthwlbkihqt
zvzbqunkojwzmholtwxkibsohshzbqsohuabltwkhjzbdvbwfbhmhsktljasjojlbkbo
zbabfkthwsohuablzMtwjaanjsojfktfjabcjlsabhwubbozjabztzvzbqkhzihrkibbmmbfktgbwbzz
hmsohshzbqlhqbajwqrbsobzbwkkibhsktlJajqgboktztwxzkojkbxtbzfhoobzshwqtwxkhqtmmbobwk
fhlsbktkthwztkvjkthwZ
```

Figure 3: Encrypted Ciphertext File



```
[Connors-MacBook-Pro:affine connor$ affine -d output.txt original.txt 11 9
```

Figure 4: Decryption Process



```
Inthispaperweconsidertheproblemofrobustfacerecognitionusingcolor
informationinthiscontextsparserepresentationbasedalgorithmsarethe
stateoftheartsolutionsforgrayfacialimageSproposedmodelthecontrolpar
ameterizationTechniquetOgetherwiththeconstrainttranscriptionmethodi
susedbytransformingtheproposedproblemintoasequenceofoptimalparameter
selectionproblemsFinallyapracticalexampleonbeersalesisusedtoshowtheeffectiveness
ofproposedmodelandwepresenttheoptimAladvertisingstrategiescorrespondingtodifferent
competitionsituationS
```

d

Figure 5: Recovered Plaintext file

Affine Mathematical Proof

The encryption and decryption functions for the affine cipher are as follows:

$$E(x) = (ax + b)\ mod\ m$$

$$D(x) = a^{-1}(x - b)\ mod\ m$$

The modular multiplicative inverse of a is defined as:

$$1 = aa^{-1}\ mod\ m$$

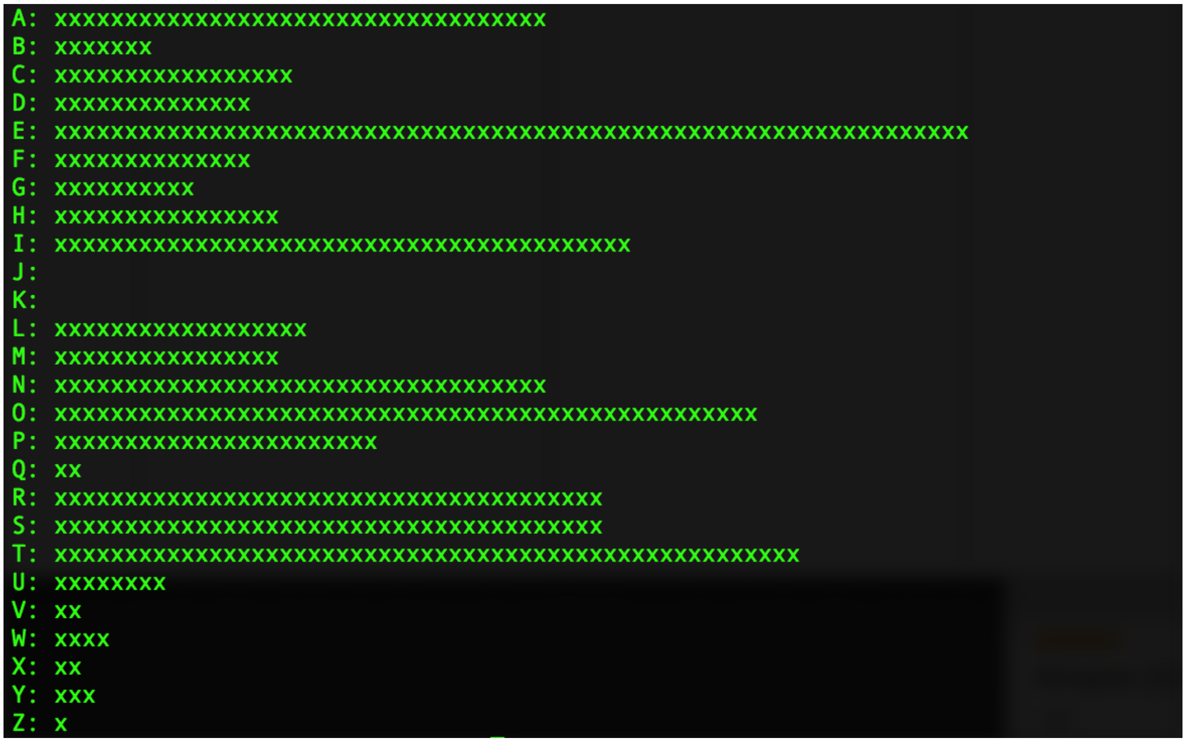
It can be shown that *D(x)* is the inverse of *E(x)* via the modular arithmetic laws.

$$\begin{aligned} D(E(x)) &= a^{-1}(E(x) - b)\ mod\ m \\ &= a^{-1}((ax + b\ mod\ m) - b)\ mod\ m \\ &= a^{-1}(ax + b - b)\ mod\ m \\ &= a^{-1}ax\ mod\ m \\ &= x\ mod\ m \end{aligned}$$

Letter Distribution

For the given test file shown in Figure 2, the following table and Figure 7 illustrate the letter distribution and relative frequencies.

- A: 35
- B: 7
- C: 17
- D: 14
- E: 65
- F: 14
- G: 10
- H: 16
- I: 41
- J: 0
- K: 0
- L: 18
- M: 16
- N: 35
- O: 50
- P: 23
- Q: 2
- R: 39
- S: 39
- T: 53
- U: 8
- V: 2
- W: 4
- X: 2
- Y: 3
- Z: 1



```
A: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
B: xxxxxxx
C: xxxxxxxxxxxxxxxxx
D: xxxxxxxxxxxxxx
E: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
F: xxxxxxxxxxxxxx
G: xxxxxxxxxx
H: xxxxxxxxxxxxxxxx
I: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
J:
K:
L: xxxxxxxxxxxxxxxxxx
M: xxxxxxxxxxxxxxxx
N: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
O: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
P: xxxxxxxxxxxxxxxxxxxxxxx
Q: xx
R: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
S: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
T: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
U: xxxxxxxx
V: xx
W: xxxx
X: xx
Y: xxx
Z: x
```

Figure 6: Letter Distributions

S-DES

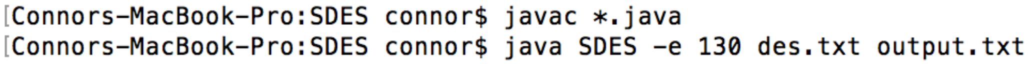
S-DES Mathematical Proof

hello

Pseudo Code Structure

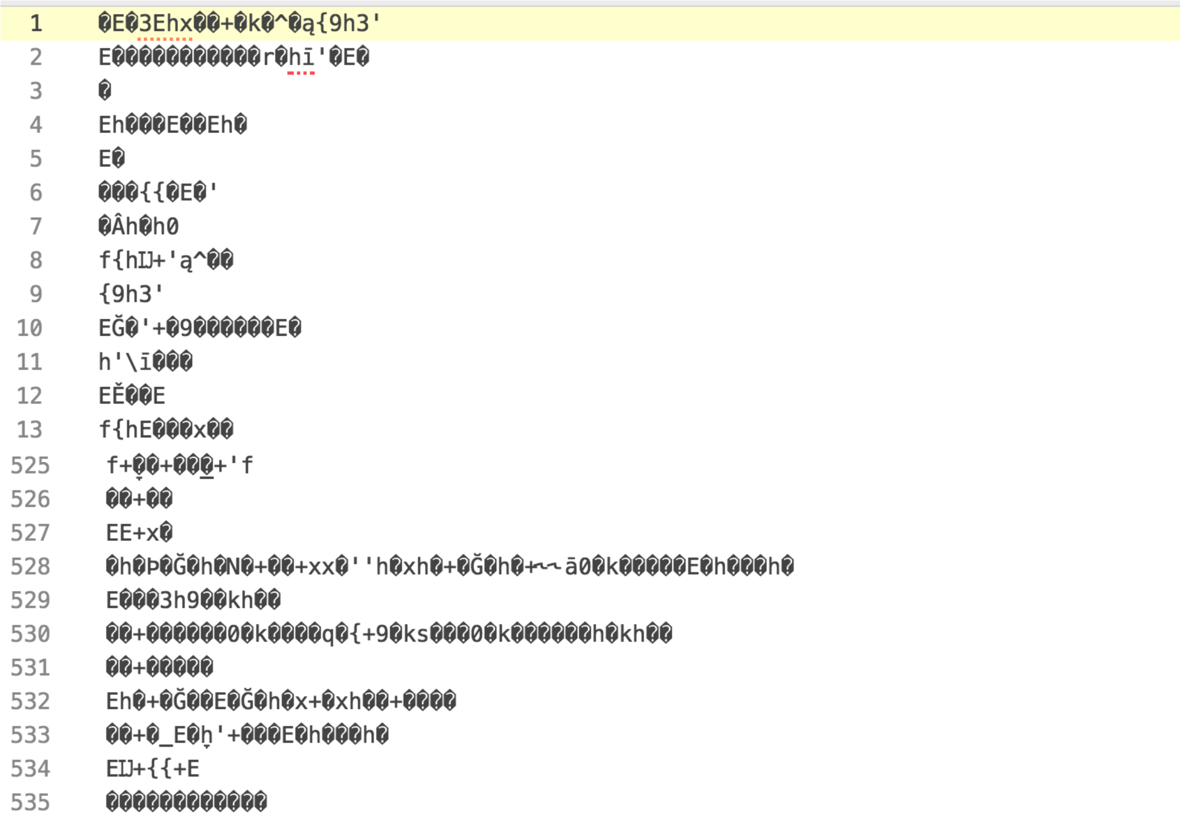
hello

Encrypted Test File



```
[Connors-MacBook-Pro:SDES connor$ javac *.java
[Connors-MacBook-Pro:SDES connor$ java SDES -e 130 des.txt output.txt
```

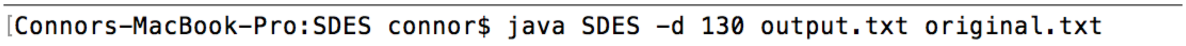
Figure 7: S-DES Encryption Process



```
1     �E�3Ehx��+�k�^�ą{9h3'
2     E������������r�hī'�E�
3     �
4     Eh���E��Eh�
5     E�
6     ���{{�E�'
7     �Âh�h0
8     f{hIJ+'ą^��
9     {9h3'
10    EĞ�'+�9�������E�
11    h'\ī���
12    EĚ��E
13    f{hE���x��
525    f+��+���+'f
526    ��+��
527    EE+x�
528    �h�Þ�Ğ�h�N�+��+xx�''h�xh�+�Ğ�h�+⌐¬ā0�k�����E�h���h�
529    E���3h9��kh��
530    ��+�������0�k����q�{+9�ks���0�k������h�kh��
531    ��+�����
532    Eh�+�Ğ��E�Ğ�h�x+�xh��+����
533    ��+�_E�ḥ'+���E�h���h�
534    EIJ+{{+E
535    �������������
```

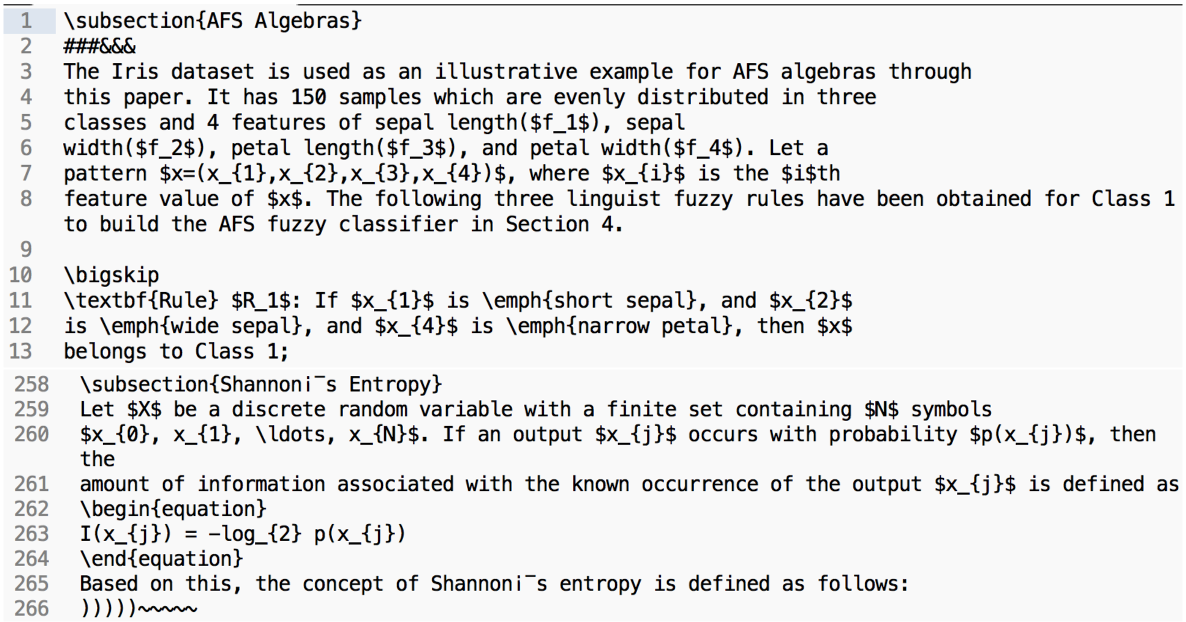
Figure 8: S-DES Cipher Text

Decrypted Test File



```
[Connors-MacBook-Pro:SDES connor$ java SDES -d 130 output.txt original.txt
```

Figure 9: S-DES Decryption Process



```
\subsection{AFS Algebras}
###&&&
The Iris dataset is used as an illustrative example for AFS algebras through
this paper. It has 150 samples which are evenly distributed in three
classes and 4 features of sepal length($f_1$), sepal
width($f_2$), petal length($f_3$), and petal width($f_4$). Let a
pattern $x=(x_{1},x_{2},x_{3},x_{4})$, where $x_{i}$ is the $i$th
feature value of $x$. The following three linguist fuzzy rules have been obtained for Class 1 to build the AFS fuzzy classifier in Section 4.

\bigskip
\textbf{Rule} $R_1$: If $x_{1}$ is \emph{short sepal}, and $x_{2}$
is \emph{wide sepal}, and $x_{4}$ is \emph{narrow petal}, then $x$
belongs to Class 1;
```

```
\subsection{Shannoni¯s Entropy}
Let $X$ be a discrete random variable with a finite set containing $N$ symbols
$x_{0}, x_{1}, \ldots, x_{N}$. If an output $x_{j}$ occurs with probability $p(x_{j})$, then the
amount of information associated with the known occurrence of the output $x_{j}$ is defined as
\begin{equation}
I(x_{j}) = -log_{2} p(x_{j})
\end{equation}
Based on this, the concept of Shannoni¯s entropy is defined as follows:
)))))~~~~~
```

Figure 10: S-DES Plain Text

Utilization of an all 1 Key

hello

Modify S-Boxes

hello

Additional Questions

Threats

hello

Source Coding

Source coding in information transmission aims to compress natural messages for highly efficient message transfer.

Error Coding

Error coding in information transmission attempts to enable a high information rate by the introduction of redundancy to data, as well as via error detection and correction mechanisms.

S-DES Coding

hello

S-DES Confusion and Diffusion

In S-DES, confusion is provided by the S-BOX substitutions performed within the feistal key round. Diffusion in contrast is provided by the permutations applied to the data included the expansion permutation utilized.

Affine Source Code

keyeligible.h

```
/****************************************************************************
* FILE: keyEligible.h
* AUTHOR: Connor Beardsmore - 15504319
* UNIT: FCC200
* PURPOSE: Header file for key eligiblility checks
*   LAST MOD: 24/03/17
*   REQUIRES: NONE
****************************************************************************/

// PROTOTYPES
int keyEligible(int,int,int);
int gcdFunction(int,int);
int extendEuclid(int,int);

//------------------------------------------------------------------------------
```

keyeligible.c

```c
/****************************************************************************
* FILE: keyEligible.c
* AUTHOR: Connor Beardsmore - 15504319
* UNIT: FCC200
* PURPOSE: Check the eligibility of keys a and b
*   LAST MOD: 28/03/17
*   REQUIRES: keyEligible.h
****************************************************************************/

#include "keyEligible.h"

//-------------------------------------------------------------------------------
// FUNCTION: keyEligible
// IMPORT: a (int), b (int)
// EXPORT: eligible (int)
// PURPOSE: Check that the two given keys are eligible via coprime check

int keyEligible( int a, int b, int alphabet )
{
    int eligible = 1;

    // a must be positive and less than the alpabet (26)
    if ( ( a < 0 ) || ( b > ( alphabet - 1 ) ) )
        eligible = 0;
    // a must be coprime to the alphabet length (26)
    if ( gcdFunction( a, alphabet ) != 1 )
        eligible = 0;
    // b must be positive and less than the alphabet (26)
    if ( ( b < 0 ) || ( b > ( alphabet - 1 ) ) )
        eligible = 0;
    return eligible;
}

//-------------------------------------------------------------------------------
// FUNCTION: gcd
// IMPORT: a (int), b (int)
// PURPOSE: Find greatest common denominator of 2 numbers

int gcdFunction( int a, int b )
{
    int quotient, residue, temp, gcd = 1;
    // SWAP ELEMENTS TO GET THE MAX
    if ( a < b )
    {
        temp = a;
        a = b;
        b = temp;
    }
    // CHECK IF EITHER NUMBER IS 0
    if ( a == 0 )   return b;
    if ( b == 0 )   return a;
    // SATISFY THE EQUATION: A = B * quotient + residue
    quotient = a / b;
    residue = a - ( b * quotient );
    // RECURSIVELY CALL GCD
    gcd = gcdFunction( b, residue );
    return gcd;
}

//-------------------------------------------------------------------------------
// FUNCTION: extendEuclid
// IMPORT: a (int), n (int)
// PURPOSE: Extended Euclidean algorithm to find inverse modular
```

```c

int extendEuclid( int a, int n )
{
    int t = 0, newt = 1;
    int r = n, newr = a;
    int q = 0, temp = 0;

    // IF GCD IS NOT 1 THEN NO COPRIME EXISTS
    if ( gcdFunction( a, n ) != 1 )
        return -1;

    // PERFORM EXTENDED EUCLIDEAN
    while ( newr != 0 )
    {
        q = r / newr;
        temp = t;
        t = newt;
        newt = temp - ( q * newt );
        temp = r;
        r = newr;
        newr = temp - ( q * newr );
    }

    // MAKE SURE T IS NOT NEGATIVE
    if ( t < 0 )
        t = t + n;

    return t;
}

//------------------------------------------------------------------------------
```

affine.h

```c
/*************************************************************************
* FILE: affine.h
* AUTHOR: Connor Beardsmore - 15504319
* UNIT: FCC200
* PURPOSE: Header file for affine cipher
*   LAST MOD: 11/03/17
*   REQUIRES: stdio.h, ctype.h, stdlib.h, string.h, keyEligible.h
*************************************************************************/

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include "keyEligible.h"

// PROTOTYPES
char encrypt(char,int,int);
char decrypt(char,int,int);

// FUNCTION POINTER
typedef char (*FuncPtr)(char,int,int);

// CONSTANTS
#define ARGS 6
#define ALPHABET 26

//---------------------------------------------------------------------------
```

affine.c

```c
/***************************************************************************
* FILE: affine.c
* AUTHOR: Connor Beardsmore - 15504319
* UNIT: FCC200
* PURPOSE: Run Affine cipher given text and key, either encrypt or decrypt
*   LAST MOD: 28/03/17
*   REQUIRES: affine.h
***************************************************************************/

#include "affine.h"

//-------------------------------------------------------------------------------
// FUNCTION: main

int main( int argc, char* argv[] )
{
    if ( argc != ARGS )
    {
        printf("\nUSAGE: <FLAG> <INPUT FILE> <OUTPUT FILE> <KEY A> <KEY B>\n");
        printf("FLAGS ARE: -e for encryption, -d for decryption\n\n");
        return 1;
    }

    // CONVERT ARGC NAMES
    char* flag = argv[1];
    char* inFile = argv[2];
    char* outFile = argv[3];
    int a = atoi( argv[4] );
    int b = atoi( argv[5] );

    // CHECK THAT THE KEYS ARE ELIGIBLE
    int validity = keyEligible( a, b, ALPHABET );
    if ( validity != 1 )
    {
        printf("\nKEYS %d AND %d ARE NOT VALID.\n", a, b );
        return 2;
    }

    // OPEN INPUT AND OUTPUT FILES
    FILE* inF = fopen( inFile, "r" );
    FILE* outF = fopen( outFile, "w" );

    // CHECK OPEN FOR ERRORS
    if ( ( inF == NULL ) || ( outF == NULL ) )
    {
        perror("\nERROR OPENING INPUT OR OUTPUT FILE\n");
        return 3;
    }

    // FUNCTION POINTER FOR encrypt() OR decrypt()
    FuncPtr fp;

    // PERFORM ENCRYPTION IF -e FLAG PROVIDED AND VICE VERSA
    if ( !strncmp( flag, "-e", 2 ) )
        fp = &encrypt;
    else if ( !strncmp( flag, "-d", 2 ) )
        fp = &decrypt;
    else
    {
        printf("\nFLAG IS INCORRECT, MUST BE -e OR -d\n");
        return 4;
    }

    // PERFROM APPROPRIATE FUNCTION
    while ( ( feof( inF ) == 0 ) && ( ferror( inF ) == 0) && (ferror( inF ) == 0) )
    {
        // GET THE NEXT CHARACTER FROM FILE
        char next = fgetc( inF );
        if ( feof( inF ) == 0 )
            // WRITE THE CONVERTED CHARACTER TO FILE
            fputc( ( *fp )( next, a, b ), outF );
    }

    // CLOSE FILES
    fclose( inF );
    fclose( outF );

    return 0;
}

//---------------------------------------------------------------------------------
// FUNCTION: encrypt
// IMPORT: plain (char), a (int), b (int)
// PURPOSE: Convert a plaintext char into the encryped character

char encrypt( char plain, int a, int b)
{
    char output = plain;
    // ENCRYPT BASED ON plain * a + b MODULO 26
    // IGNORE NON-CHARACTERS
    if ( isupper(plain) )
        output = ( ( ( plain - 'A' ) * a + b ) % ALPHABET ) + 'A';
    else if ( islower(plain) )
        output = ( ( ( plain - 'a' ) * a + b ) % ALPHABET ) + 'a';
    return output;
}

//---------------------------------------------------------------------------------
// FUNCTION: decrypt
// IMPORT: plain (char*), a (int), b (int)
// PURPOSE: Convert a ciphertect char into the decrypted character

char decrypt( char cipher, int a, int b )
{
    // FIND THE MODULO INVERSE USING EUCLIDEAN
    int inverse = extendEuclid( a, ALPHABET );
    char output = cipher;
    // DECRYPT BASED ON inverse * cipher - b MODULO 26
    // IGNORE NON-CHARACTERS
    if ( isupper(cipher) )
        output = ( ( inverse * ( cipher - 'A' - b + ALPHABET ) ) % ALPHABET ) + 'A';
    else if ( islower(cipher) )
        output = ( ( inverse * ( cipher - 'a' - b + ALPHABET ) ) % ALPHABET ) + 'a';
    return output;
}

//---------------------------------------------------------------------------------
```

S-DES Source Code

SDESConstants.java

```java
/***************************************************************************
* FILE: SDESConstants
* AUTHOR: Connor Beardsmore - 15504319
* UNIT: FCC200
* PURPOSE: Structures to represent the constants in the SDES algorithm
*   LAST MOD: 21/03/17
*   REQUIRES: NONE
***************************************************************************/

public class SDESConstants
{
    // P10 PERMUTATION FOR THE 10-BIT KEY
    public static final int[] P10 = { 2, 4, 1, 6, 3, 9, 0, 8, 7, 5 };

//-----------------------------------------------------------------------------

    // P8 PERMUTATION FOR THE 10-BIT KEY
    public static final int[] P8 = { 5, 2, 6, 3, 7, 4, 9, 8 };

//-----------------------------------------------------------------------------

    // INITIAL PERMUTATION FOR THE 8-BIT PLAINTEXT
    public static final int[] IP = { 1, 5, 2, 0, 3, 7, 4, 6 };

//-----------------------------------------------------------------------------

    // INVERSE PERMUTATION FOR THE 8-BIT PLAINTEXT
    public static final int[] IPI = { 3, 0, 2, 4, 6, 1, 7, 5 };

//-----------------------------------------------------------------------------

    // EXPANSION PERMUTATION FOR 4-BITS IN Fk
    public static final int[] EP = { 3, 0, 1, 2, 1, 2, 3, 0 };

//-----------------------------------------------------------------------------

    // P4 PERMUTATION AFTER THE S-BOX SELECTION
    public static final int[] P4 = { 1, 3, 2, 0 };

//-----------------------------------------------------------------------------

    // SBOX ONE
    public static final int[][] S0 = {  { 1, 0, 3, 2 },
                                        { 3, 2, 1, 0 },
                                        { 0, 2, 1, 3 },
                                        { 3, 1, 3, 2 }  };

//-----------------------------------------------------------------------------

    // SBOX TWO
    public static final int[][] S1 = {  { 0, 1, 2, 3 },
                                        { 2, 0, 1, 3 },
                                        { 3, 0, 1, 0},
                                        { 2, 1, 0, 3 }  };

//-----------------------------------------------------------------------------
}
```

SDESBits.java

```
/***************************************************************************
* FILE: SDESBits.java
* AUTHOR: Connor Beardsmore − 15504319
* UNIT: FCC200
* PURPOSE: BitSet alternative using a int
*   LAST MOD: 24/03/17
*   REQUIRES: NONE
***************************************************************************/

public class SDESBits
{
    //CONSTANTS
    public static final int MIN_SIZE = 4;
    public static final int MAX_SIZE = 10;

    //CLASSFIELDS
    private int bits;
    private int size;
    private int half;
    // private only applies to different classes, so we can
    // import an SDESBits and retreive bits without a getter

//-----------------------------------------------------------------------------
    //ALTERNATE CONSTRUCTOR

    public SDESBits( int inBits, int inSize )
    {
        // Check inBits and inSize validity
        if ( inBits < 0 )
            throw new IllegalArgumentException( ”INVALID SDESBits VALUE” );
        if ( ( inSize < MIN_SIZE ) || ( inSize > MAX_SIZE ) )
            throw new IllegalArgumentException( ”INVALID SDESBits SIZE” );
        if ( inSize % 2 != 0 )
            throw new IllegalArgumentException( ”INVALID SDESBits SIZE” );

        bits = inBits;
        size = inSize;
        half = inSize >>> 1;
    }

//-----------------------------------------------------------------------------
    //COPY CONSTRUCTOR

    public SDESBits( SDESBits inBits )
    {
        bits = inBits.bits;
        size = inBits.size;
        half = inBits.half;
    }

//-----------------------------------------------------------------------------
    //FUNCTION: switchHalves()
    //PURPOSE: Switch the left half of bits with the right half

    public void switchHalves()
    {
        // Get the right half of the bits
        int oRight = bits & ( ( 1 << half ) − 1 );
        // Shift the left half of the bits down
        bits >>>= half;
        // Combine left half with right half shifted up
        bits |= ( oRight << half );
    }
```

```

//------------------------------------------------------------------------------
    //FUNCTION: permute()
    //IMPORT: permTable (int[])
    //EXPORT: permuted (SDESBits)
    //PURPOSE: Create a permutation of this objects bits in a new SDESBits

    public SDESBits permute( int[] permTable )
    {
        // Create temporary space the size of the permutation
        SDESBits permuted = new SDESBits( 0, permTable.length );
        // Iterate across the permutation, getting and setting bits
        for ( int ii = 0; ii < permTable.length; ii++ )
            permuted.setBit( getBit( permTable[ii] ), ii );
        return permuted;
    }

//------------------------------------------------------------------------------
    //FUNCTION: leftShift()
    //IMPORT: shifts (int)
    //PURPOSE: Perfrom a circular left shift on the bits of each half

    public void leftShift( int shifts )
    {
        //Check shift validity
        if ( shifts < 1 )
            throw new IllegalArgumentException( "ILLEGAL SHIFT VALUE" );

        // Temp variable for repeated 1's for a half
        int ones = ( 1 << half ) - 1;
        // Avoid shifting more than required
        if ( half > shifts )
            shifts %= half;

        // Get the left half and right half
        int left = bits >>> half;
        int right = bits & ones;

        // Loop for each shift individually
        for ( int ii = 0; ii < shifts; ii++ )
        {
            // Get the leftmost bit of the left sub-half
            int leftBit = ( left & ones );
            leftBit >>>= MIN_SIZE;
            // Get the rightmost bit of the right sub-half
            int rightBit = ( right & ones );
            rightBit >>>= MIN_SIZE;

            // Perform the actual shifting of the bits
            left = ( left << 1 ) & ones;
            right = ( right << 1 ) & ones;

            // If the first bits of the halves were one, set final bit
            if ( leftBit == 1 )      left++;
            if ( rightBit == 1 )     right++;
        }

        // Recombine both halves back together
        bits = ( left << half ) | right;
    }

//------------------------------------------------------------------------------
    //FUNCTION: split()
    //EXPORT: halves (SDESBits[])
    //PURPOSE: Split the bits into two sub-halves and return as objects

    public SDESBits[] split()
    {
        // New container for the halves
        SDESBits[] halves = new SDESBits[2];
        // Get the left half and create object
        int leftInt = bits >>> half;
        halves[0] = new SDESBits( leftInt, half );
        // Get the right half and create object
        int rightInt = ( bits & ( ( 1 << half ) - 1 ) );
        halves[1] = new SDESBits( rightInt, half );

        return halves;
    }

//------------------------------------------------------------------------------
    //FUNCTION: xor()
    //IMPORT: inBits (SDESBits)
    //PURPOSE: XOR bits with the bits value of inBits

    public void xor( SDESBits inBits )
    {
        // Ensure the same size
        if ( size != inBits.size )
            throw new IllegalArgumentException( "CANNOT XOR DIFFERENT SIZES" );

        // Call simple exclusive-or on both bits
        bits ^= inBits.bits;
    }

//------------------------------------------------------------------------------
    //FUNCTION: setBit()
    //IMPORT: val (boolean), index (int)
    //PURPOSE: Set the value at the specifed index with the specified value

    public void setBit( boolean val, int index )
    {
        // Validity
        if ( ( index < 0 ) || ( index >= size) )
            throw new IllegalArgumentException("SETBIT IMPORTS INVALID");

        // Reset the given bit
        bits &= ~(1 << ( size - index - 1 ) );
        // Reset the bits greater than the size we want
        bits &= (1 << size)-1;
        // Set the required bit
        bits |= ((val) ? 1 : 0 ) << ( size - index - 1 );
    }

//------------------------------------------------------------------------------
    //FUNCTION: getBit()
    //IMPORT: index (int)
    //EXPORT: value (boolean)
    //PURPOSE: Get the value of the bit at the specified index

    public boolean getBit( int index )
    {
        if ( ( index < 0 ) || ( index >= size) )
            throw new IllegalArgumentException("SETBIT IMPORTS INVALID");

        // Bits are reverse ordered
        return (bits & 1 << ( size - index - 1 ) ) != 0;
    }

//------------------------------------------------------------------------------
```

```java

    public int getBits() { return bits; }

//---------------------------------------------------------------------------
    //FUNCTION: append()
    //IMPORT: newBits (SDESBits)
    //PURPOSE: Append new set of bits to the original set

    public void append( SDESBits newBits )
    {
        // Increment size
        size += newBits.size;
        // Shift original across and add new bits
        bits = ( bits << newBits.size ) | newBits.bits;
        // Update half value
        half = size >>> 1;
    }

//---------------------------------------------------------------------------
    //FUNCTION: sbox()
    //EXPORT: result (int)
    //PURPOSE: Find the sbox values for the bits in this object

    public int sbox()
    {
        // Split into halves
        SDESBits halves[] = this.split();

        // Get row and column of the first four bits
        int colS0 = ( halves[0].bits & 6 ) >>> 1;
        int rowS0 = ( ( halves[0].bits & 8 ) >>> 2 ) | ( halves[0].bits & 1 );
        // Get row and column of the second four bits
        int colS1 = ( halves[1].bits & 6 ) >>> 1;
        int rowS1 = ( ( halves[1].bits & 8 ) >>> 2 ) | ( halves[1].bits & 1 );

        // Get the appropriate sbox value
        int s0Val = SDESConstants.S0[rowS0][colS0];
        int s1Val = SDESConstants.S1[rowS1][colS1];

        // Combine the result
        int result = ( s0Val << 2 ) | s1Val;

        return result;
    }

//---------------------------------------------------------------------------
    //FUNCTION: toString()
    //EXPORT: state (String)
    //PURPOSE: Export bits in a readable binary format

    public String toString()
    {
        return Integer.toBinaryString( bits );
    }

//---------------------------------------------------------------------------
}
```

SDES.java

```java
/***************************************************************************
* FILE: SDES.java
* AUTHOR: Connor Beardsmore - 15504319
* UNIT: FCC200
* PURPOSE: Performs SDES encryption or decryption on a given file
*   LAST MOD: 21/03/17
*   REQUIRES: NONE
***************************************************************************/

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

public class SDES
{

    public static final int NUM_ARGS = 4;
    public static final int MAX_KEY = 1023;
    public static final int KEY_SIZE = 10;
    public static final int MESSAGE_SIZE = 8;

//---------------------------------------------------------------------------

    public static void main( String[] args )
    {
        // Check argument length and output usage
        if ( args.length != NUM_ARGS )
        {
            System.out.println("USAGE: SDES <mode> <key> <input file> <output file>");
            System.out.println("modes = -e encryption, -d decryption");
            System.out.println("keys = int between 0 and 255");
            System.exit(1);
        }

        // Rename variables for simplicity
        String mode = args[0];
        String key = args[1];
        String inFile = args[2];
        String outFile = args[3];
        SDESBits message, output;
        int intKey = Integer.parseInt( key );

        try
        {
            // Generate subkeys
            SDESBits subkeys[] = keyGeneration( intKey );

            // Open file streams
            FileInputStream fis = new FileInputStream( new File( inFile ) );
            FileOutputStream fos = new FileOutputStream( new File( outFile ) );

            // Read bytes until end of file
            int next = fis.read();
            while ( next != -1 )
            {
                message = new SDESBits( next, MESSAGE_SIZE );

                // Select function based on mode
                if ( mode.equals( "-e" ) )
                    output = encrypt( message, subkeys  );
                else if ( mode.equals( "-d") )
                    output = decrypt( message, subkeys  );
                else
                    throw new IllegalArgumentException("INVALID MODE");

                // Write converted output to file
                int outputInt = output.getBits();
                fos.write( outputInt );
                next = fis.read();
            }
        }
        catch (Exception e)
        {
            System.out.println( e.getMessage() );
        }

    }

//---------------------------------------------------------------------------
    //FUNCTION: encrypt()
    //IMPORT: message (SDESBits), subkeys (SDESBits[])
    //EXPORT: message (SDESBits)
    //PURPOSE: Encrypt given message with given subkeys

    public static SDESBits encrypt( SDESBits message, SDESBits[] subkeys )
    {
        // Initial Permutation
        message = message.permute( SDESConstants.IP );
        // First feistal key round with subkey 1
        message = feistalRound( message, subkeys[0] );
        // Switch left and right subhalves
        switchFunction( message );
        // Second feistal key round with subkey 2
        message = feistalRound( message, subkeys[1] );
        // Inverse of Initial Permutation
        message = message.permute( SDESConstants.IPI );
        return message;
    }

//---------------------------------------------------------------------------
    //FUNCTION: decrypt()
    //IMPORT: message (SDESBits), subkeys (SDESBits[])
    //EXPORT: message (SDESBits)
    //PURPOSE: Decrypt given message with given subkeys

    public static SDESBits decrypt( SDESBits message, SDESBits[] subkeys )
    {
        // Initial Permutation
        message = message.permute( SDESConstants.IP );
        // First feistal key round with subkey 2
        message = feistalRound( message, subkeys[1] );
        // Switch left and right subhalves
        switchFunction( message );
        // First feistal key round with subkey 2
        message = feistalRound( message, subkeys[0] );
        // Inverse of Initial Permutation
        message = message.permute( SDESConstants.IPI );
        return message;
    }

//---------------------------------------------------------------------------
    //FUNCTION: switchFunction()
    //IMPORT: input (SDESBitSet)
    //PURPOSE: Import 8-bit binary and swap the first and last 4 bits

    public static void switchFunction( SDESBits input )
    {
        input.switchHalves();
    }
```

```java

//---------------------------------------------------------------------------------
    //FUNCTION: keyGeneration()
    //IMPORT: keyDec (int)
    //EXPORT: subkeys (SDESBits[])
    //PURPOSE: Generate subkeys given the full key

    public static SDESBits[] keyGeneration( int keyDec )
    {
        // Check key validity
        if ( ( keyDec < 0 ) || ( keyDec > MAX_KEY ) )
            throw new IllegalArgumentException(”INVALID KEY”);

        // Convert int key into an SDESBits object and create subkey array
        SDESBits key = new SDESBits( keyDec, KEY_SIZE );
        SDESBits[] subkeys = new SDESBits[2];

        // P10 permutation, left shift and P8 permutation to form subkey 1
        key = key.permute( SDESConstants.P10 );
        key.leftShift(1);
        subkeys[0] = key.permute( SDESConstants.P8 );
        // P8 permutation and double left shift to form subkey 2
        key.leftShift(2);
        subkeys[1] = key.permute( SDESConstants.P8 );

        return subkeys;
    }

//---------------------------------------------------------------------------------
    //FUNCTION: feistalRound()
    //IMPORT: message (SDESBits), subkey (SDESBits)
    //EXPORT: halves (SDESBits)
    //PURPOSE: Perform feistal key round on message given a subkey

    public static SDESBits feistalRound( SDESBits message, SDESBits subkey )
    {
        // Split message in half
        SDESBits halves[] = message.split();
        // Perform fMapping function
        SDESBits fMap = fMapping( halves[1], subkey );
        // XOR the halves and append
        halves[0].xor( fMap );
        halves[0].append(halves[1]);
        return halves[0];
    }

//---------------------------------------------------------------------------------
    //FUNCTION: fMapping()
    //IMPORT: message (SDESBits), subkey (SDESBits)
    //EXPORT: message (SDESBits)
    //PURPOSE: Perform fMapping function on given message with subkey

    public static SDESBits fMapping( SDESBits message, SDESBits subkey )
    {
        // Expansion permutation and XOR with subkey
        message = message.permute( SDESConstants.EP );
        message.xor( subkey );
        // Calculate SBOX values and P4 permutation
        message = new SDESBits( message.sbox(), MESSAGE_SIZE/2 );
        message = message.permute( SDESConstants.P4 );
        return message;
    }

//---------------------------------------------------------------------------------
}
```