

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.

```

1 //-----
2 // FUNCTION: gcd
3 // IMPORT: a (int), b (int)
4 // PURPOSE: Find greatest common denominator of 2 numbers
5
6 int gcdFunction( int a, int b )
7 {
8     int quotient, residue, temp, gcd = 1;
9     // SWAP ELEMENTS TO GET THE MAX
10    if ( a < b )
11    {
12        temp = a;
13        a = b;
14        b = temp;
15    }
16    // CHECK IF EITHER NUMBER IS 0
17    if ( a == 0 ) return b;
18    if ( b == 0 ) return a;
19    // SATISFY THE EQUATION: A = B * quotient + residue
20    quotient = a / b;
21    residue = a - ( b * quotient );
22    // RECURSIVELY CALL GCD
23    gcd = gcdFunction( b, residue );
24    return gcd;
25 }
26
27 //-----

```

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

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

The full list of valid b values is: **0 to 25 inclusive**

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 (Stallings 2011).

Recovered Plaintext

The implemented Affine cipher works correctly with all eligible keys. The recovered plaintext in Figure 3 is identical to the original plaintext of Figure 1. No major problems were encountered during the implementation phase.

```

1  Inthispaperweconsidertheproblemofrobustfacerecognitionusingcolor
2  informationinthiscontextsparserepresentationbasedalgorithmsarethe
3  stateoftheartsolutionsforgrayfacialimageSproposedmodelthecontrolpar
4  ameterizationTechniquetOgetherwiththeconstrainttranscriptionmethodi
5  susedbytransformingtheproposedproblemintoasequenceofoptimalparameter
6  selectionproblemsFinallyapracticalexampleonbeersalesisusedtoshowtheeffectiveness
7  ofproposedmodelandwepresenttheoptimaladvertisingstrategiescorrespondingtodifferent
8  competitionsituationS
9

```

Figure 1: Original Plaintext

```

1  Twkitzsjborbfhwztqbokibsohuablmohuvzkmjfbobfhxwtkthwvztwxfhaho
2  twmholjktwtkitzfhwkbczsjozbobsobzbwkjktwujzbqjaxhotkilzjobkib
3  zkjbhmkibjokzhavkthwzmhoxojnmjftjatljxbZsohshzbqlhqbakibfhwkohasjo
4  jlbkbotyjkthwKbfiwtdvbKHXbkibortkikibfhwzkojtwkkojwzfotskthwlbkiht
5  zvzbqunkojwzmholtwxkibsohshzbqsohuabltwkhjzbdvbwfbhmhsktljasjojlbkbo
6  zbabfkthwsouablzMtwjaanjsojfktfjabcjlsabhwubbozjabztzvzbqkhzihrkibbmbfktgbwbzz
7  hmsohshzbqlhqabajwqrsobzbwkibhsktlJajqgboktztwxzkojkbxtbzfoobzshwqtwxkhqtmmbobwk
8  fhlsbktkthwztkvjktwZ
9

```

Figure 2: Encrypted Ciphertext

```

1  Inthispaperweconsidertheproblemofrobustfacerecognitionusingcolor
2  informationinthiscontextsparserepresentationbasedalgorithmsarethe
3  stateoftheartsolutionsforgrayfacialimageSproposedmodelthecontrolpar
4  ameterizationTechniquetOgetherwiththeconstrainttranscriptionmethodi
5  susedbytransformingtheproposedproblemintoasequenceofoptimalparameter
6  selectionproblemsFinallyapracticalexampleonbeersalesisusedtoshowtheeffectiveness
7  ofproposedmodelandwepresenttheoptimaladvertisingstrategiescorrespondingtodifferent
8  competitionsituationS
9

```

Figure 3: Recovered Plaintext

Affine Mathematical Proof

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

$$E(x) = c = (ax + b) \bmod m$$

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

where:

$$x = \textit{plaintext}$$

$$c = \textit{ciphertext}$$

$$m = \textit{alphabet length}$$

$$a, b = \textit{keys}$$

The modular multiplicative inverse of a - a^{-1} - is defined as:

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

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

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

Letter Distribution

For the given test file shown in Figure 2, the following values and Figure 4 illustrate the letter distribution and relative frequencies. The breakdown of the relative frequencies allow for cryptanalysis to be performed on the affine cipher, thus leading to a factor in its insecurity.

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

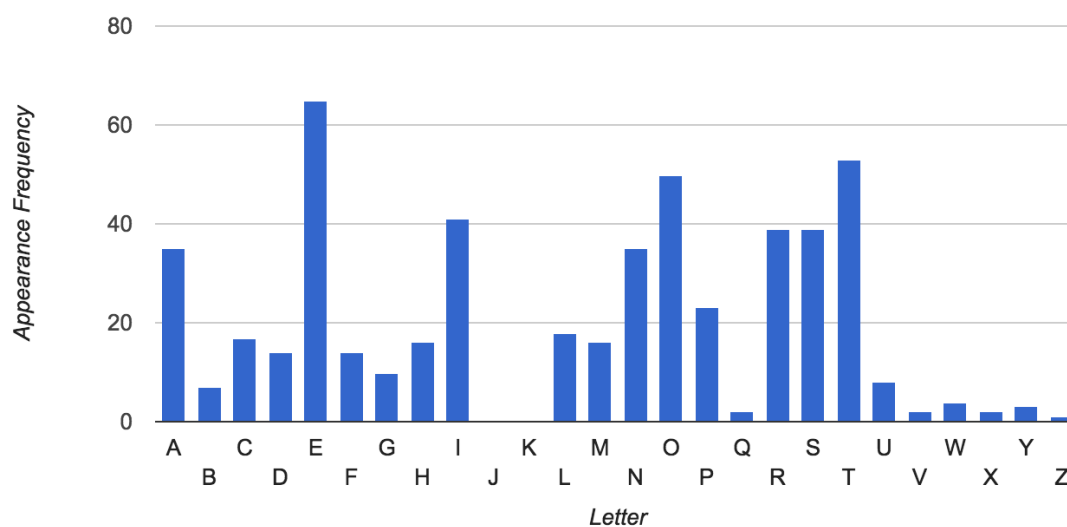


Figure 4: Letter Distributions

S-DES

S-DES Mathematical Proof

Let the following definitions apply:

$m = 8bit\ plaintext$

$c = 8bit\ ciphertext$

$k = 10bit\ key$

$IP = initial\ permutation$

$IP^{-1} = inverse\ of\ initial\ permutation$

$L_i = left\ block\ of\ m\ (5bit)$

$R_i = right\ block\ of\ m\ (5bit)$

Let encryption be defined as the following set of three steps:

Step 1

$$(L_0, R_0) = IP(p)$$

Step 2

Let $i = 1, 2$

Let $k_i = subkey\ i\ (8bit)$

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, k_i)$$

Step 3

$$c = IP^{-1}(L_{16}, R_{16})$$

Next, it can be proven that plaintext m can be discovered from ciphertext c given the full key k :

$$(L'_0, R'_0) = IP(m)$$

$$\therefore (L'_0, R'_0) = (R_2, L_2)$$

$$R'_0 = [L_1 \oplus f(R_1, k_2)] \oplus [f(R_1, k_2)]$$

$$R'_1 = L_1$$

$$L'_1 = R_1$$

Pseudo Code Structure

The pseudo-code structure of the three key functions utilized in the S-DES implementation is illustrated below. The three functions constitute the core functionality of the S-DES algorithm.

```
function KEYGENERATION(int key)
  key  $\leftarrow$  PERMUTE( key, P10 )
  LEFTSHIFT( key, 1 )
  subkeys[0]  $\leftarrow$  PERMUTE( key, P8 )
  LEFTSHIFT( key, 2 )
  subkeys[1]  $\leftarrow$  PERMUTE( key, P8 )
  return subkeys
end function
```

```
function SWITCHFUNCTION(int input)
  right  $\leftarrow$  bits && ((1 << 4) - 1)
  left  $\leftarrow$  bits >>> 4
  output  $\leftarrow$  left || (right << 4)
  return output
end function
```

```
function FEISTALKEYROUND(int message, int subkey)
  halves  $\leftarrow$  SPLIT( message )
  fMap  $\leftarrow$  FMAPPING( rightHalf, subkey )
  leftHalf  $\leftarrow$  leftHalf  $\oplus$  fMap
  return combined leftHalf + rightHalf
end function
```

For clarity, the pseudo code structure for the fMap function in addition to how the above three functions are combined for encryption is covered below.

```
function FMAPPING( int message, int subkey)
  message  $\leftarrow$  PERMUTE( message, EP )
  message  $\leftarrow$  message  $\oplus$  subkey
  message  $\leftarrow$  SBOX_CALCULATION(message)
  message  $\leftarrow$  PERMUTE( message, P4 )
  return message
end function
```

```
function ENCRYPTION( int message, int[] subkey)
  message  $\leftarrow$  PERMUTE( message, IP )
  message  $\leftarrow$  FEISTALROUND( message, subkey[0] )
  SWITCHFUNCTION(message)
  message  $\leftarrow$  FEISTALROUND( message, subkey[1] )
  message  $\leftarrow$  PERMUTE( message, IP_INVERSE )
  return message
end function
```


Figure 5 illustrates the overall scheme of the simplified DES algorithm. It is evident that the algorithm is symmetric due to decryption being the clear inverse of encryption. The key generation scheme is also displayed.

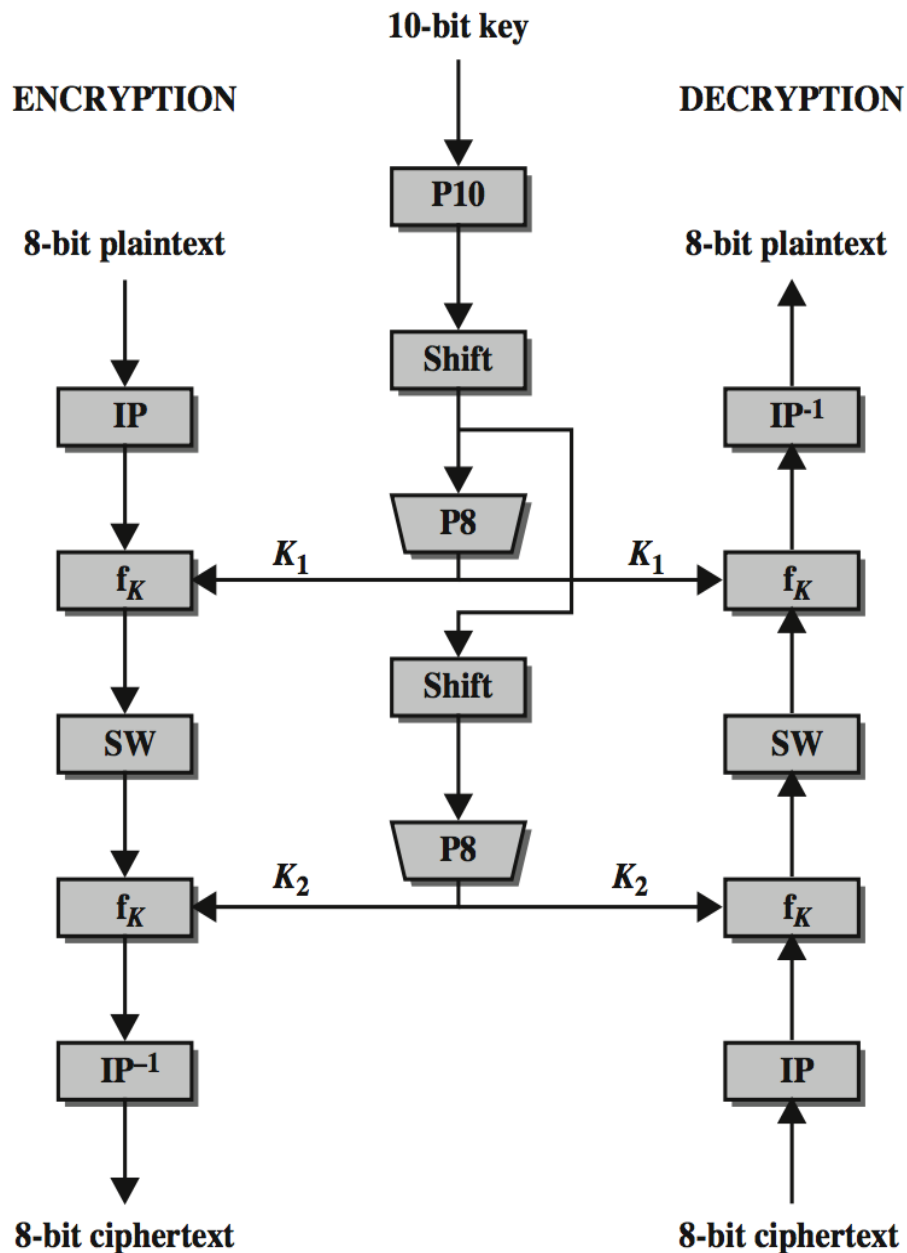


Figure 5: Simplified DES Scheme (Stallings 2011)

Encrypted Test File

The included code implementation for S-DES works correctly and there were no difficulties faced in the process of programming. Figure 6 shows the plaintext after encryption while Figure 7 produces the original plaintext via decryption of the ciphertext.

```

1  0E03Ehx00+0k0^0a{9h3'
2  E00000000000r0h1'0E0
3  0
4  Eh000E00Eh0
5  E0
6  000-{0E0'
7  0Ãh0h0
8  f{hI+ 'a^00
9  {9h3'
10 EĜ0'+09000000E0
11 h'\i000
12 EÊ00E
13 f{hE000x00
525 f+00+000+' f
526 00+00
527 EE+x0
528 0h0P0Ĝ0h0N0+00+xx0' 'h0xh0+0Ĝ0h0+~ã00k00000E0h000h0
529 E0003h900kh00
530 00+00000000k0000q0{+90ks00000k000000h0kh00
531 00+00000
532 Eh0+0Ĝ00E0Ĝ0h0x+0xh00+0000
533 00+0_E0h'+000E0h000h0
534 EI+{-{+E
535 000000000000

```

Figure 6: S-DES Cipher Text

Decrypted Test File

```

1  \subsection{AFS Algebras}
2  ###&&&
3  The Iris dataset is used as an illustrative example for AFS algebras through
4  this paper. It has 150 samples which are evenly distributed in three
5  classes and 4 features of sepal length($f_1$), sepal
6  width($f_2$), petal length($f_3$), and petal width($f_4$). Let a
7  pattern $x=(x_{\{1\}},x_{\{2\}},x_{\{3\}},x_{\{4\}})$, where $x_{\{i\}}$ is the $i$th
8  feature value of $x$. The following three linguist fuzzy rules have been obtained for Class 1 to build the
9
257
258 \subsection{Shannon's Entropy}
259 Let $X$ be a discrete random variable with a finite set containing $N$ symbols
260 $x_{\{0\}}, x_{\{1\}}, \ldots, x_{\{N\}}$. If an output $x_{\{j\}}$ occurs with probability $p(x_{\{j\}})$, then the
261 amount of information associated with the known occurrence of the output $x_{\{j\}}$ is defined as
262 \begin{equation}
263 I(x_{\{j\}}) = -\log_{\{2\}} p(x_{\{j\}})
264 \end{equation}
265 Based on this, the concept of Shannon's entropy is defined as follows:
266 )))))~~~~~

```

Figure 7: S-DES Plain Text

Utilization of an all 1 Key

Performing encryption and decryption with S-DES utilizing a key of all 1's (11111111) does not significantly alter how the algorithm performs. However, during the two feistel key rounds, the subkeys will be equivalent. The S-DES algorithm is symmetric, with the difference between encryption and decryption being the different subkeys utilized in the two rounds (Konikoff and Toplosky 2010). If the subkeys are identical, both encryption and decryption end up being the same process. Thus in this situation, both the following apply:

$$x = E(E(x))$$

$$x = D(D(x))$$

The same situation will occur with a key of all 0's (0000000000) or a key of alternating 1's and 0's (1010101010) for the same reasons. These keys are termed "weak keys" (Schneier 1996). Due to the large keyspace of S-DES, the rare occurrence of these weak keys is not considered a major flaw in the overriding algorithm.

Modify S-Boxes

The S-BOX values in the SDESConstants.java file were modified to ensure that the S-DES algorithm still performs accurately. Modification of the S-BOX values within the given constraints - each row must contain 0, 1, 2 and 3 - did not affect the overall algorithm. However, modification of the S-BOX values can reduce the security of DES significantly (Stallings 2011).

Gargiulo 2002 discusses S-BOX design in-depth, particularly how their design permits the DES algorithm to ensure the "Strict Avalanche Criteria". The majority of attacks on the DES algorithm are centred about the S-BOXES and potential vulnerabilities within them.

Additional Questions

Threats

There are numerous type of threats that exist within information transmission. These types consist of both errors with data transmission and specific security attacks performed by a third party. In any channel, there exists error causing effects such as noise and interference (Liu 2017). These effects cannot be completely prevented in wireless channels and thus, error detection and correction mechanisms must be employed.

In addition to errors that can occur within data channels, security threats are also prevalent within information transmission. Security attacks aim to compromise one of the three fundamental components of information security: confidentiality, integrity and availability. Two forms of security attacks are possible, either passive or attack. There are infinite forms of security attack, with common examples including denial of service, replay attack, man in the middle and session hijacking.

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. A simplistic example of error coding is the repetition code. Forms of error detection and correction mechanisms include parity checksums in addition to the ISBN and UPC codes.

S-DES Coding

The source coding in the S-DES implementation involves converting the plaintext ascii key into a 10-bit binary code. This is done by taking the hash code of the initial String and performing chopping to ensure a maximum 10-bit key. The implementation shown reads each byte individually from the file and thus no source coding was required. However if the file was read as ascii characters, the conversion of this into a binary representation would be a form of source coding.

The S-DES algorithm does not contain any error coding. No checkbits or replication bits are included to help detect or correct errors. This is in contrast to the standard DES algorithm, where 8-bits of the 64-bit key are odd parity checksums. This implies that S-DES has a higher information rate than DES, at the cost of the possible detection of transmission errors (Schneier 1996).

S-DES Confusion and Diffusion

In S-DES, both of Shannon's key principles - confusion and diffusion - are present in order to achieve security. Confusion is the property where each bit of ciphertext depends upon multiple parts of the key (Stallings 2011). This property is provided in S-DES by the S-BOX substitutions performed within the feistel key round.

Diffusion is the property where a single bit change in the plaintext should respect in multiple bit changes in the ciphertext. This property is provided by the permutations applied to the data included the expansion permutation utilized.

Affine Source Code

keyeligible.h

```
1  /*****
2  * FILE: keyEligible.h
3  * AUTHOR: Connor Beardsmore - 15504319
4  * UNIT: FCC200
5  * PURPOSE: Header file for key eligibility checks
6  *   LAST MOD: 24/03/17
7  *   REQUIRES: NONE
8  *****/
9
10 // PROTOTYPES
11 int keyEligible(int, int, int);
12 int gcdFunction(int, int);
13 int extendEuclid(int, int);
14
15 //
```

keyeligible.c

```

1  /*****
2  * FILE: keyEligible.c
3  * AUTHOR: Connor Beardsmore - 15504319
4  * UNIT: FCC200
5  * PURPOSE: Check the eligibility of keys a and b
6  *   LAST MOD: 28/03/17
7  *   REQUIRES: keyEligible.h
8  *****/
9
10 #include "keyEligible.h"
11
12 //-----
13 // FUNCTION: keyEligible
14 // IMPORT: a (int), b (int)
15 // EXPORT: eligible (int)
16 // PURPOSE: Check that the two given keys are eligible via coprime check
17
18 int keyEligible( int a, int b, int alphabet )
19 {
20     int eligible = 1;
21
22     // a must be positive and less than the alphabet (26)
23     if ( ( a < 0 ) || ( b > ( alphabet - 1 ) ) )
24         eligible = 0;
25     // a must be coprime to the alphabet length (26)
26     if ( gcdFunction( a, alphabet ) != 1 )
27         eligible = 0;
28     // b must be positive and less than the alphabet (26)
29     if ( ( b < 0 ) || ( b > ( alphabet - 1 ) ) )
30         eligible = 0;
31     return eligible;
32 }
33
34 //-----
35 // FUNCTION: gcd
36 // IMPORT: a (int), b (int)
37 // PURPOSE: Find greatest common denominator of 2 numbers
38
39 int gcdFunction( int a, int b )
40 {
41     int quotient, residue, temp, gcd = 1;
42     // SWAP ELEMENTS TO GET THE MAX
43     if ( a < b )
44     {
45         temp = a;
46         a = b;
47         b = temp;
48     }
49     // CHECK IF EITHER NUMBER IS 0
50     if ( a == 0 ) return b;
51     if ( b == 0 ) return a;
52     // SATISFY THE EQUATION: A = B * quotient + residue
53     quotient = a / b;
54     residue = a - ( b * quotient );
55     // RECURSIVELY CALL GCD
56     gcd = gcdFunction( b, residue );
57     return gcd;
58 }
59
60 //-----
61 // FUNCTION: extendEuclid
62 // IMPORT: a (int), n (int)
63 // PURPOSE: Extended Euclidean algorithm to find inverse modular

```

```
64
65 int extendEuclid( int a, int n )
66 {
67     int t = 0, newt = 1;
68     int r = n, newr = a;
69     int q = 0, temp = 0;
70
71     // IF GCD IS NOT 1 THEN NO COPRIME EXISTS
72     if ( gcdFunction( a, n ) != 1 )
73         return -1;
74
75     // PERFORM EXTENDED EUCLIDEAN
76     while ( newr != 0 )
77     {
78         q = r / newr;
79         temp = t;
80         t = newt;
81         newt = temp - ( q * newt );
82         temp = r;
83         r = newr;
84         newr = temp - ( q * newr );
85     }
86
87     // MAKE SURE T IS NOT NEGATIVE
88     if ( t < 0 )
89         t = t + n;
90
91     return t;
92 }
93
94 //
```

affine.h

```
1  /*****
2  * FILE: affine.h
3  * AUTHOR: Connor Beardsmore - 15504319
4  * UNIT: FCC200
5  * PURPOSE: Header file for affine cipher
6  *   LAST MOD: 11/03/17
7  *   REQUIRES: stdio.h, ctype.h, stdlib.h, string.h, keyEligible.h
8  *****/
9
10 #include <stdio.h>
11 #include <ctype.h>
12 #include <stdlib.h>
13 #include <string.h>
14 #include "keyEligible.h"
15
16 // PROTOTYPES
17 char encrypt(char, int, int);
18 char decrypt(char, int, int);
19
20 // FUNCTION POINTER
21 typedef char (*FuncPtr)(char, int, int);
22
23 // CONSTANTS
24 #define ARGS 6
25 #define ALPHABET 26
26
27 //
```


affine.c

```

1  /*****
2  * FILE: affine.c
3  * AUTHOR: Connor Beardsmore - 15504319
4  * UNIT: FCC200
5  * PURPOSE: Run Affine cipher given text and key, either encrypt or decrypt
6  *   LAST MOD: 28/03/17
7  *   REQUIRES: affine.h
8  *****/
9
10 #include "affine.h"
11
12 //-----
13 // FUNCTION: main
14
15 int main( int argc, char* argv[] )
16 {
17     if ( argc != ARGS )
18     {
19         printf("\nUSAGE: <FLAG> <INPUT FILE> <OUTPUT FILE> <KEY A> <KEY B>\n");
20         printf("FLAGS ARE: -e for encryption, -d for decryption\n\n");
21         return 1;
22     }
23
24     // CONVERT ARGV NAMES
25     char* flag = argv[1];
26     char* inFile = argv[2];
27     char* outFile = argv[3];
28     int a = atoi( argv[4] );
29     int b = atoi( argv[5] );
30
31     // CHECK THAT THE KEYS ARE ELIGIBLE
32     int validity = keyEligible( a, b, ALPHABET );
33     if ( validity != 1 )
34     {
35         printf("\nKEYS %d AND %d ARE NOT VALID.\n", a, b );
36         return 2;
37     }
38
39     // OPEN INPUT AND OUTPUT FILES
40     FILE* inF = fopen( inFile, "r" );
41     FILE* outF = fopen( outFile, "w" );
42
43     // CHECK OPEN FOR ERRORS
44     if ( ( inF == NULL ) || ( outF == NULL ) )
45     {
46         perror( "\nERROR OPENING INPUT OR OUTPUT FILE\n" );
47         return 3;
48     }
49
50     // FUNCTION POINTER FOR encrypt() OR decrypt()
51     FuncPtr fp;
52
53     // PERFORM ENCRYPTION IF -e FLAG PROVIDED AND VICE VERSA
54     if ( !strcmp( flag, "-e", 2 ) )
55         fp = &encrypt;
56     else if ( !strcmp( flag, "-d", 2 ) )
57         fp = &decrypt;
58     else
59     {
60         printf("\nFLAG IS INCORRECT, MUST BE -e OR -d\n");
61         return 4;
62     }
63

```

```

64 // PERFORM APPROPRIATE FUNCTION
65 while ( ( feof( inF ) == 0 ) && ( ferror( inF ) == 0 ) && ( ferror( inF ) == 0 ) )
66 {
67     // GET THE NEXT CHARACTER FROM FILE
68     char next = fgetc( inF );
69     if ( feof( inF ) == 0 )
70         // WRITE THE CONVERTED CHARACTER TO FILE
71         fputc( ( *fp )( next, a, b ), outF );
72 }
73
74 // CLOSE FILES
75 fclose( inF );
76 fclose( outF );
77
78 return 0;
79 }
80
81 //-----
82 // FUNCTION: encrypt
83 // IMPORT: plain (char), a (int), b (int)
84 // PURPOSE: Convert a plaintext char into the encrypted character
85
86 char encrypt( char plain, int a, int b )
87 {
88     char output = plain;
89     // ENCRYPT BASED ON plain * a + b MODULO 26
90     // IGNORE NON-CHARACTERS
91     if ( isupper(plain) )
92         output = ( ( ( plain - 'A' ) * a + b ) % ALPHABET ) + 'A';
93     else if ( islower(plain) )
94         output = ( ( ( plain - 'a' ) * a + b ) % ALPHABET ) + 'a';
95     return output;
96 }
97
98 //-----
99 // FUNCTION: decrypt
100 // IMPORT: plain (char*), a (int), b (int)
101 // PURPOSE: Convert a ciphertext char into the decrypted character
102
103 char decrypt( char cipher, int a, int b )
104 {
105     // FIND THE MODULO INVERSE USING EUCLIDEAN
106     int inverse = extendEuclid( a, ALPHABET );
107     char output = cipher;
108     // DECRYPT BASED ON inverse * cipher - b MODULO 26
109     // IGNORE NON-CHARACTERS
110     if ( isupper(cipher) )
111         output = ( ( inverse * ( cipher - 'A' - b + ALPHABET ) ) % ALPHABET ) + 'A';
112     else if ( islower(cipher) )
113         output = ( ( inverse * ( cipher - 'a' - b + ALPHABET ) ) % ALPHABET ) + 'a';
114     return output;
115 }
116
117 //-----

```

S-DES Source Code

SDESConstants.java

```

1  /*****
2  * FILE: SDESConstants
3  * AUTHOR: Connor Beardsmore - 15504319
4  * UNIT: FCC200
5  * PURPOSE: Structures to represent the constants in the SDES algorithm
6  *   LAST MOD: 21/03/17
7  *   REQUIRES: NONE
8  *****/
9
10 public class SDESConstants
11 {
12     // P10 PERMUTATION FOR THE 10-BIT KEY
13     public static final int[] P10 = { 2, 4, 1, 6, 3, 9, 0, 8, 7, 5 };
14
15     //-----
16
17     // P8 PERMUTATION FOR THE 10-BIT KEY
18     public static final int[] P8 = { 5, 2, 6, 3, 7, 4, 9, 8 };
19
20     //-----
21
22     // INITIAL PERMUTATION FOR THE 8-BIT PLAINTEXT
23     public static final int[] IP = { 1, 5, 2, 0, 3, 7, 4, 6 };
24
25     //-----
26
27     // INVERSE PERMUTATION FOR THE 8-BIT PLAINTEXT
28     public static final int[] IPI = { 3, 0, 2, 4, 6, 1, 7, 5 };
29
30     //-----
31
32     // EXPANSION PERMUTATION FOR 4-BITS IN Fk
33     public static final int[] EP = { 3, 0, 1, 2, 1, 2, 3, 0 };
34
35     //-----
36
37     // P4 PERMUTATION AFTER THE S-BOX SELECTION
38     public static final int[] P4 = { 1, 3, 2, 0 };
39
40     //-----
41
42     // SBOX ONE
43     public static final int[][] S0 = {
44         { 1, 0, 3, 2 },
45         { 3, 2, 1, 0 },
46         { 0, 2, 1, 3 },
47         { 3, 1, 3, 2 }
48     };
49
50     // SBOX TWO
51     public static final int[][] S1 = {
52         { 0, 1, 2, 3 },
53         { 2, 0, 1, 3 },
54         { 3, 0, 1, 0 },
55         { 2, 1, 0, 3 }
56     };
57 }

```

SDESBits.java

```

1  /*****
2  * FILE: SDESBits.java
3  * AUTHOR: Connor Beardsmore - 15504319
4  * UNIT: FCC200
5  * PURPOSE: BitSet alternative using a int
6  *   LAST MOD: 24/03/17
7  *   REQUIRES: NONE
8  *****/
9
10 public class SDESBits
11 {
12     //CONSTANTS
13     public static final int MIN_SIZE = 4;
14     public static final int MAX_SIZE = 10;
15
16     //CLASSFIELDS
17     private int bits;
18     private int size;
19     private int half;
20     // private only applies to different classes, so we can
21     // import an SDESBits and retrieve bits without a getter
22
23     //-----
24     //ALTERNATE CONSTRUCTOR
25
26     public SDESBits( int inBits , int inSize )
27     {
28         // Check inBits and inSize validity
29         if ( inBits < 0 )
30             throw new IllegalArgumentException( "INVALID SDESBits VALUE" );
31         if ( ( inSize < MIN_SIZE ) || ( inSize > MAX_SIZE ) )
32             throw new IllegalArgumentException( "INVALID SDESBits SIZE" );
33         if ( inSize % 2 != 0 )
34             throw new IllegalArgumentException( "INVALID SDESBits SIZE" );
35
36         bits = inBits;
37         size = inSize;
38         half = inSize >>> 1;
39     }
40
41     //-----
42     //COPY CONSTRUCTOR
43
44     public SDESBits( SDESBits inBits )
45     {
46         bits = inBits.bits;
47         size = inBits.size;
48         half = inBits.half;
49     }
50
51     //-----
52     //FUNCTION: switchHalves()
53     //PURPOSE: Switch the left half of bits with the right half
54
55     public void switchHalves()
56     {
57         // Get the right half of the bits
58         int oRight = bits & ( ( 1 << half ) - 1 );
59         // Shift the left half of the bits down
60         bits >>= half;
61         // Combine left half with right half shifted up
62         bits |= ( oRight << half );
63     }

```

```

64
65 //-----
66 //FUNCTION: permute()
67 //IMPORT: permTable (int[])
68 //EXPORT: permuted (SDESBits)
69 //PURPOSE: Create a permutation of this objects bits in a new SDESBits
70
71 public SDESBits permute( int[] permTable )
72 {
73     // Create temporary space the size of the permutation
74     SDESBits permuted = new SDESBits( 0, permTable.length );
75     // Iterate across the permutation, getting and setting bits
76     for ( int ii = 0; ii < permTable.length; ii++ )
77         permuted.setBit( getBit( permTable[ii] ), ii );
78     return permuted;
79 }
80
81 //-----
82 //FUNCTION: leftShift()
83 //IMPORT: shifts (int)
84 //PURPOSE: Perform a circular left shift on the bits of each half
85
86 public void leftShift( int shifts )
87 {
88     //Check shift validity
89     if ( shifts < 1 )
90         throw new IllegalArgumentException( "ILLEGAL SHIFT VALUE" );
91
92     // Temp variable for repeated 1's for a half
93     int ones = ( 1 << half ) - 1;
94     // Avoid shifting more than required
95     if ( half > shifts )
96         shifts %= half;
97
98     // Get the left half and right half
99     int left = bits >>> half;
100    int right = bits & ones;
101
102    // Loop for each shift individually
103    for ( int ii = 0; ii < shifts; ii++ )
104    {
105        // Get the leftmost bit of the left sub-half
106        int leftBit = ( left & ones );
107        leftBit >>>= MIN_SIZE;
108        // Get the rightmost bit of the right sub-half
109        int rightBit = ( right & ones );
110        rightBit >>>= MIN_SIZE;
111
112        // Perform the actual shifting of the bits
113        left = ( left << 1 ) & ones;
114        right = ( right << 1 ) & ones;
115
116        // If the first bits of the halves were one, set final bit
117        if ( leftBit == 1 )    left++;
118        if ( rightBit == 1 )   right++;
119    }
120
121    // Recombine both halves back together
122    bits = ( left << half ) | right;
123 }
124
125 //-----
126 //FUNCTION: split()
127 //EXPORT: halves (SDESBits[])
128 //PURPOSE: Split the bits into two sub-halves and return as objects

```

```

129
130     public SDESBits[] split()
131     {
132         // New container for the halves
133         SDESBits[] halves = new SDESBits[2];
134         // Get the left half and create object
135         int leftInt = bits >>> half;
136         halves[0] = new SDESBits( leftInt , half );
137         // Get the right half and create object
138         int rightInt = ( bits & ( ( 1 << half ) - 1 ) );
139         halves[1] = new SDESBits( rightInt , half );
140
141         return halves;
142     }
143
144 //-----
145 //FUNCTION: xor()
146 //IMPORT: inBits (SDESBits)
147 //PURPOSE: XOR bits with the bits value of inBits
148
149     public void xor( SDESBits inBits )
150     {
151         // Ensure the same size
152         if ( size != inBits.size )
153             throw new IllegalArgumentException( "CANNOT XOR DIFFERENT SIZES" );
154
155         // Call simple exclusive-or on both bits
156         bits ^= inBits.bits;
157     }
158
159 //-----
160 //FUNCTION: setBit()
161 //IMPORT: val (boolean), index (int)
162 //PURPOSE: Set the value at the specified index with the specified value
163
164     public void setBit( boolean val , int index )
165     {
166         // Validity
167         if ( ( index < 0 ) || ( index >= size ) )
168             throw new IllegalArgumentException("SETBIT IMPORTS INVALID");
169
170         // Reset the given bit
171         bits &= ~(1 << ( size - index - 1 ) );
172         // Reset the bits greater than the size we want
173         bits &= (1 << size)-1;
174         // Set the required bit
175         bits |= ((val) ? 1 : 0 ) << ( size - index - 1 );
176     }
177
178 //-----
179 //FUNCTION: getBit()
180 //IMPORT: index (int)
181 //EXPORT: value (boolean)
182 //PURPOSE: Get the value of the bit at the specified index
183
184     public boolean getBit( int index )
185     {
186         if ( ( index < 0 ) || ( index >= size ) )
187             throw new IllegalArgumentException("SETBIT IMPORTS INVALID");
188
189         // Bits are reverse ordered
190         return (bits & 1 << ( size - index - 1 ) ) != 0;
191     }
192
193 //-----

```

```
194
195     public int getBits() { return bits; }
196
197 //-----
198 //FUNCTION: append()
199 //IMPORT: newBits (SDESBits)
200 //PURPOSE: Append new set of bits to the original set
201
202     public void append( SDESBits newBits )
203     {
204         // Increment size
205         size += newBits.size;
206         // Shift original across and add new bits
207         bits = ( bits << newBits.size ) | newBits.bits;
208         // Update half value
209         half = size >>> 1;
210     }
211
212 //-----
213 //FUNCTION: sbbox()
214 //EXPORT: result (int)
215 //PURPOSE: Find the sbbox values for the bits in this object
216
217     public int sbbox()
218     {
219         // Split into halves
220         SDESBits halves[] = this.split();
221
222         // Get row and column of the first four bits
223         int colS0 = ( halves[0].bits & 6 ) >>> 1;
224         int rowS0 = ( ( halves[0].bits & 8 ) >>> 2 ) | ( halves[0].bits & 1 );
225         // Get row and column of the second four bits
226         int colS1 = ( halves[1].bits & 6 ) >>> 1;
227         int rowS1 = ( ( halves[1].bits & 8 ) >>> 2 ) | ( halves[1].bits & 1 );
228
229         // Get the appropriate sbbox value
230         int s0Val = SDESConstants.S0[rowS0][colS0];
231         int s1Val = SDESConstants.S1[rowS1][colS1];
232
233         // Combine the result
234         int result = ( s0Val << 2 ) | s1Val;
235
236         return result;
237     }
238
239 //-----
240 //FUNCTION: toString()
241 //EXPORT: state (String)
242 //PURPOSE: Export bits in a readable binary format
243
244     public String toString()
245     {
246         return Integer.toBinaryString( bits );
247     }
248
249 //-----
250 }
```

SDES.java

```

1  /*****
2  * FILE: SDES.java
3  * AUTHOR: Connor Beardsmore - 15504319
4  * UNIT: FCC200
5  * PURPOSE: Performs SDES encryption or decryption on a given file
6  *   LAST MOD: 21/03/17
7  *   REQUIRES: NONE
8  *****/
9
10 import java.util.*;
11 import java.io.*;
12
13 public class SDES
14 {
15
16     public static final int NUMARGS = 4;
17     public static final int MAXKEY = 1023;
18     public static final int KEY_SIZE = 10;
19     public static final int MESSAGE_SIZE = 8;
20
21     //-----
22
23     public static void main( String[] args )
24     {
25         // Check argument length and output usage
26         if ( args.length != NUMARGS )
27         {
28             System.out.println("USAGE: SDES <mode> <key> <input file> <output file>");
29             System.out.println("modes = -e encryption, -d decryption");
30             System.out.println("keys = int between 0 and 255");
31             System.exit(1);
32         }
33
34         // Rename variables for simplicity
35         String mode = args[0];
36         String key = args[1];
37         String inFile = args[2];
38         String outFile = args[3];
39         SDESBits message, output;
40
41         int intKey = createKey( key );
42
43         try
44         {
45             // Generate subkeys
46             SDESBits subkeys[] = keyGeneration( intKey );
47
48             // Open file streams
49             FileInputStream fis = new FileInputStream( new File( inFile ) );
50             FileOutputStream fos = new FileOutputStream( new File( outFile ) );
51
52             // Read bytes until end of file
53             int next = fis.read();
54             while ( next != -1 )
55             {
56                 message = new SDESBits( next, MESSAGE_SIZE );
57
58                 // Select function based on mode
59                 if ( mode.equals( "-e" ) )
60                     output = encrypt( message, subkeys );
61                 else if ( mode.equals( "-d" ) )
62                     output = decrypt( message, subkeys );
63                 else

```



```

64         throw new IllegalArgumentException("INVALID MODE");
65
66         // Write converted output to file
67         int outputInt = output.getBits();
68         fos.write( outputInt );
69         next = fis.read();
70     }
71 }
72 catch (Exception e)
73 {
74     System.out.println( e.getMessage() );
75 }
76
77 }
78
79 //-----
80 //FUNCTION: encrypt()
81 //IMPORT: message (SDESBits), subkeys (SDESBits[])
82 //EXPORT: message (SDESBits)
83 //PURPOSE: Encrypt given message with given subkeys
84
85 public static SDESBits encrypt( SDESBits message, SDESBits[] subkeys )
86 {
87     // Initial Permutation
88     message = message.permute( SDESConstants.IP );
89     // First feistel key round with subkey 1
90     message = feistelRound( message, subkeys[0] );
91     // Switch left and right subhalves
92     switchFunction( message );
93     // Second feistel key round with subkey 2
94     message = feistelRound( message, subkeys[1] );
95     // Inverse of Initial Permutation
96     message = message.permute( SDESConstants.IPI );
97     return message;
98 }
99
100 //-----
101 //FUNCTION: decrypt()
102 //IMPORT: message (SDESBits), subkeys (SDESBits[])
103 //EXPORT: message (SDESBits)
104 //PURPOSE: Decrypt given message with given subkeys
105
106 public static SDESBits decrypt( SDESBits message, SDESBits[] subkeys )
107 {
108     // Initial Permutation
109     message = message.permute( SDESConstants.IP );
110     // First feistel key round with subkey 2
111     message = feistelRound( message, subkeys[1] );
112     // Switch left and right subhalves
113     switchFunction( message );
114     // First feistel key round with subkey 1
115     message = feistelRound( message, subkeys[0] );
116     // Inverse of Initial Permutation
117     message = message.permute( SDESConstants.IPI );
118     return message;
119 }
120
121 //-----
122 //FUNCTION: switchFunction()
123 //IMPORT: input (SDESBitSet)
124 //PURPOSE: Import 8-bit binary and swap the first and last 4 bits
125
126 public static void switchFunction( SDESBits input )
127 {
128     input.switchHalves();

```

```

129     }
130
131 //-----
132 //FUNCTION: createKey()
133 //IMPORT: key (String)
134 //EXPORT: intKey (int)
135 //PURPOSE: Convert string key from user into a valid 10-bit integer
136
137 public static int createKey( String key )
138 {
139     // Chop to ensure it's only 10-bit long in total
140     // No need to pad if we do this chopping correctly
141     return ( key.hashCode() & MAXKEY );
142 }
143
144 //-----
145 //FUNCTION: keyGeneration()
146 //IMPORT: keyDec (int)
147 //EXPORT: subkeys (SDESBits[])
148 //PURPOSE: Generate subkeys given the full key
149
150 public static SDESBits[] keyGeneration( int keyDec )
151 {
152     // Check key validity
153     if ( ( keyDec < 0 ) || ( keyDec > MAXKEY ) )
154         throw new IllegalArgumentException("INVALID KEY");
155
156     // Convert int key into an SDESBits object and create subkey array
157     SDESBits key = new SDESBits( keyDec, KEY_SIZE );
158     SDESBits[] subkeys = new SDESBits[2];
159
160     // P10 permutation, left shift and P8 permutation to form subkey 1
161     key = key.permute( SDESConstants.P10 );
162     key.leftShift(1);
163     subkeys[0] = key.permute( SDESConstants.P8 );
164     // P8 permutation and double left shift to form subkey 2
165     key.leftShift(2);
166     subkeys[1] = key.permute( SDESConstants.P8 );
167
168     return subkeys;
169 }
170
171 //-----
172 //FUNCTION: feistalRound()
173 //IMPORT: message (SDESBits), subkey (SDESBits)
174 //EXPORT: halves (SDESBits)
175 //PURPOSE: Perform feistal key round on message given a subkey
176
177 public static SDESBits feistalRound( SDESBits message, SDESBits subkey )
178 {
179     // Split message in half
180     SDESBits halves[] = message.split();
181     // Perform fMapping function
182     SDESBits fMap = fMapping( halves[1], subkey );
183     // XOR the halves and append
184     halves[0].xor( fMap );
185     halves[0].append(halves[1]);
186     return halves[0];
187 }
188
189 //-----
190 //FUNCTION: fMapping()
191 //IMPORT: message (SDESBits), subkey (SDESBits)
192 //EXPORT: message (SDESBits)
193 //PURPOSE: Perform fMapping function on given message with subkey

```

```
194
195     public static SDESBits fMapping( SDESBits message, SDESBits subkey )
196     {
197         // Expansion permutation and XOR with subkey
198         message = message.permute( SDESConstants.EP );
199         message.xor( subkey );
200         // Calculate SBOX values and P4 permutation
201         message = new SDESBits( message.sbox(), MESSAGE_SIZE/2 );
202         message = message.permute( SDESConstants.P4 );
203         return message;
204     }
205
206 //-----
207 }
```

References

- Gargiulo, Joe. 2002. "S-Box Modifications and Their Effect in DES-like Encryption Systems". *SANS Institute InfoSec Reading Room*.
- Konikoff, Jacob, and Seth Toplosky. 2010. "Analysis of Simplified DES Algorithms". *Cryptologia* 34 (3): 211–224.
- Liu, Wan-Quan. 2017. *Lecture 3: Coding*. Curtin University.
- Schneier, Bruce. 1996. *Applied Cryptography*. 5th ed. John Wiley & Sons Inc.
- Stallings, William. 2011. *Cryptography and Network Security: Principles and Practice*. 5th ed. Prentice Hall.