Fundamental Concepts of Cryptography

Assignment 1

Jeremy Shade – 15182706

18/04/2016

Contents

[Affine Cipher 2](#_Toc449034467)

[Mathematical Proof for the Affine Cipher: 2](#_Toc449034468)

[Computing all possible eligible keys: 2](#_Toc449034469)

[Test File Input/Output: 3](#_Toc449034470)

[Test File Input: 3](#_Toc449034471)

[Test File Decrypted Output: 3](#_Toc449034472)

[Input File Character Graph: 3](#_Toc449034473)

[DES 4](#_Toc449034474)

[Mathematical Proof of DES 4](#_Toc449034475)

[Pseudo Code: 5](#_Toc449034476)

[Key Generation 5](#_Toc449034477)

[Switch Function 6](#_Toc449034478)

[Feistel Function (Fk) 6](#_Toc449034479)

[Implementation Difficulties (but all working now!): 6](#_Toc449034480)

[Appendix A – Code For DES and Affine Ciphers 6](#_Toc449034481)

[Appendix B – Output from DES Cipher 6](#_Toc449034482)

# Affine Cipher

## Mathematical Proof for the Affine Cipher:

For to exist, *a* and *n* must be coprime. Without this, the decryption may not be possible as more than one unique value of *a* may exist. Following on from this, it can be shown that the decryption function is the inverse of the encryption function:

For all letters in the standard alphabet, this yields the original character when mod 26 (as characters range from 0 to 25) and thus proves the decryption algorithm holds.

## Computing all possible eligible keys:

The possible keys generated below were **not** just for letters, but all symbols in the ascii table (ie, Dec 0 to Dec 127). This means my upper limit is equal to 128.

Like *b*, *a* has a limit where it must be less than the length of the number of characters to account for, however *a* also has the limit where it must be coprime with m, the number of characters. As such, the greatest common divisor for both a and m must be equal to *n* as seen below:

Possible keys for *a*, with n=128:

1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 4

3, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 8

3, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103, 105, 107, 109, 111, 113, 115, 117,

119, 121, 123, 125, 127.

Thus, the limits for b are

Possible keys for *b*, with *n* = 128:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22

, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42

, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62

, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82

, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101,

102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117,

118, 119, 120, 121, 122, 123, 124, 125, 126, 127

Thus, the total number of possible keys:

## Test File Input/Output:

### Test File Input:

Inthispaperweconsidertheproblemofrobustfacerecognitionusingcolor

informationinthiscontextsparserepresentationbasedalgorithmsarethe

stateoftheartsolutionsforgrayfacialimageSproposedmodelthecontrolpar

ameterizationTechniquetOgetherwiththeconstrainttranscriptionmethodi

susedbytransformingtheproposedproblemintoasequenceofoptimalparameter

selectionproblemsFinallyapracticalexampleonbeersalesisusedtoshowtheeffectiveness

ofproposedmodelandwepresenttheoptimAladvertisingstrategiescorrespondingtodifferent

competitionsituationS

### Test File Decrypted Output:

Inthispaperweconsidertheproblemofrobustfacerecognitionusingcolor

informationinthiscontextsparserepresentationbasedalgorithmsarethe

stateoftheartsolutionsforgrayfacialimageSproposedmodelthecontrolpar

ameterizationTechniquetOgetherwiththeconstrainttranscriptionmethodi

susedbytransformingtheproposedproblemintoasequenceofoptimalparameter

selectionproblemsFinallyapracticalexampleonbeersalesisusedtoshowtheeffectiveness

ofproposedmodelandwepresenttheoptimAladvertisingstrategiescorrespondingtodifferent

competitionsituationS

### Input File Character Graph:

Now the graph:

Now the graph:

Char | Count |

..................................................

A | 1 | #

F | 1 | #

I | 1 | #

O | 1 | #

S | 2 | # #

T | 1 | #

a | 34 | # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

b | 7 | # # # # # # #

c | 17 | # # # # # # # # # # # # # # # # #

d | 14 | # # # # # # # # # # # # # #

e | 65 | # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

f | 13 | # # # # # # # # # # # # #

g | 10 | # # # # # # # # # #

h | 16 | # # # # # # # # # # # # # # # #

i | 40 | # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

l | 18 | # # # # # # # # # # # # # # # # # #

m | 16 | # # # # # # # # # # # # # # # #

n | 35 | # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

o | 49 | # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

p | 23 | # # # # # # # # # # # # # # # # # # # # # # #

q | 2 | # #

r | 39 | # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

s | 37 | # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

t | 52 | # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

u | 8 | # # # # # # # #

v | 2 | # #

w | 4 | # # # #

x | 2 | # #

y | 3 | # # #

z | 1 | #

..................................................

# DES

## Mathematical Proof of DES

The operation of DES for ***encryption*** can be generalized to three steps:

\*Let Initial Permutation be represented by IP

Encryption and decryption of an input block share the same three steps above, only the iteration round direction is reversed due to key arrangement.

The arrangement of keys, also called the key schedule, is denoted by:

The ***decryption*** algorithm starts by inputting the cipher text as the input block:

However, since the input block is actually the output from the final step of encryption, it can be rewritten as:

Following on from step 2a of encryption, we deduce:

The right hand sides of these two assignments should be replaced with their corresponding iteration, I.e.:

with due to step 2a of encryption,

replaced by due to step 2b,

by due to the key schedule

Following this replacement yields steps 2a and 2b directly above as:

Therefore, after one round of decryption we acquire:

Thus, at the start of the next round the two half blocks are .

Following on, the next 15 rounds we will obtain:

For the final step, the two half blocks from the final round ( 16 ) are swapped, and input to the inverse of the initial permutation:

It can be seen that the output block above is the original input from step 1 of encryption.

Following the rule that for a decryption key,, and an encryption key, , that the following equation must be met for a cryptographic system:

That is,

Therefore, it has been shown that for DES that the encryption and decryption algorithms keep to this equation and holds true for .

## Pseudo Code:

### Key Generation

*IMPORT*: inKey

*EXPORT*: keyArray

*ALGORITHM:*

Make new byte array named keyArray of size 16, 0

tempKey = CALL permute 🡨 inKey, permChoice1

C = CALL getBits 🡨 tempKey, 0, permChoice1.length/2

D = CALL getBits 🡨 tempKey, permChoice1.length/2, permChoice1.length/2

FOR ii equals 0 to 15 do

C = CALL rotateLeft 🡨 C, 28, keyShift at ii

D = CALL rotateLeft 🡨 D, 28, keyShift at ii

cd = CALL mergeBits 🡨 C, 28, D 28

set keyArray at ii = CALL permute 🡨 cd, permChoice2

ENDFOR

### Switch Function

*IMPORT*: L, R, inKey, decrypt, ii

*EXPORT*: L, R

*ALGORITHM:*

SET tempR = R

IF decrypt is equal to TRUE do

R = CALL feistel 🡨 R, inKey at 15-ii

ELSE do

R = CALL fiestel 🡨 R, inKey at ii

ENDIF

SET R = CALL xor 🡨 L, R

SET L = tempR

### Feistel Function (Fk)

*IMPORT:* R, KEY, eTable

*EXPORT*: feistelBlock

*ALGORITHM:*

SET feistelBlock = CALL permute 🡨 R, eTable

SET feistelBlock = CALL xor 🡨 feistelBlock, KEY

SET feistelBlock = CALL shift 🡨 feistelBlock

SET feistelBlock = CALL permute 🡨 feistelBlock, feistelPerm

## Implementation Difficulties (but all working now!):

The main difficulty faced during implementation of DES was with bit shifting/bit wise ANDing as well as the use of byte arrays. Working with this caused confusion. The second biggest issue I faced was the automatic encoding of the importing and exporting of files to the hard disk. Reading in lines as strings caused erroneous and erratic artefacts to occur in the decrypted file, but once importing byte arrays and exporting byte arrays, the decrypted text adjusted to what was expected as an inverse encryption of the input text. Overall, it ended up working well.

## Key of all Zeroes:

Having changed the key from a random string of characters and numbers to a string of all 0’s yielded the same decrypted file output as expected, however the encrypted file was vastly different with regards to the symbols observed. It was expected that the encrypted output differ every time the key was changed but that the decrypted value (with the same key) would yield the same output as the input text. This was observed for varying keys tested.

Note – the key used for the encrypted text found at the end of this document was a key of all 0’s.

# Appendix A – Code For DES and Affine Ciphers

# Appendix B – Output from DES Cipher