STUDY AND IMPLEMENTATION OF VARIOUS CRYPTOGRAPHIC TECHNIQUES

SUBMITTED IN PARTIAL FULFILLMENT OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

Enrollment Number - 091022, 091042, 091076

Name of Students - Shreyas Dighe, Tushar Sharma, Saurav Dash

Name of Supervisor - Professor Dr. T. S. Lamba

Dedicated to the spirit of HACKING

```
####    ####            ###             #########      ##     #
####    ####           #####           ###  #####     ###      ##
####    ####          #######         ###    ###     ####    ###
####    ####         #########       ###      #      ####  ####
####    ####        ###########     ###               ### ####
############       ####     ####    ###               #######
############      #####     #####    ###     ##      #######
############     ######  #########    ###   ####     #########
####    ####      ######## #####       ##  ######     ###   ###
####    ####       ####     ####        #########     ###  #####
####    ####        ###     ###          #######     ####  ######
####    ####         ##     ##            #####      ####   ####
####    ####         ##     ##             ###        ###    ##
```

[1]

1 Hacking is learning and building constructing things and nothing else

ABSTRACT

In early times people used metals for trade. Gold and silver were among the most important assest which people use to hoard. After industrialization, oil became the most valuable asset. Today oil producing nations are among the richest nations. Early 90's was the period when "the walls came down and windows came up".

The fall of Berlin wall and Soviet Union with the dominant rise of Windows powered PCs encroached the landscape of a common man. With the invention of intenet by US Defence and WWW by Tim Lee, the bridge between people virtually became non-existent. Companies like `Google, Facebook, Amazon, etc` mint billions by selling information of their users. Information has become the most important commodity. Thus it become imperative to safeguard information. Cryptography is a secret science of obfuscating information.

In midst of our the semester, we studied various Cryptographic techniques. We were successful in implementing some of the following SYMMETRIC CRYPTOGRAPHIC algorithms used in both wired and wireless communication.

DES	in C++
IDEA	in C++
AES	in C++
RC4	in Python

We claim all our RESULTS are without any plagiarism. We have not tested our programs against all TEST CASES. We have tried to give all due credits to the work of any author or source we have included in our report.

ACKNOWLEDGEMENTS

We would like express our gratitude to Professor Dr.*T S Lamba* for spending his valuable time in guiding us through out our project. Without his efforts and motivations we would not have succeded in this project.

We are also grateful to the whole LaTeX community for support and ideas.

CONTENTS

LIST OF FIGURES

ACRONYMS

DRY Don't Repeat Yourself

API Application Programming Interface

UML Unified Modeling Language

1 OVERVIEW

The only secure computer is one
that's unplugged, locked in a safe,
and buried 20 feet under the
ground in a secret location...and
I'm not even too sure about that one

— Attributed Dennis Huges, FBI

Definition CRYPTOGRAPHY, CRYPTOLOGY
Cryptography comes from Greek with *crypto* means "hidden, secret" and *graphy* means "writing". Cryptography can be defined as the conversion of data into a scrambled code that can be deciphered and sent across a public or private network[1]. A broader definition is cryptology with Greek "-logy" means "science".[2]

Definition CODE
Code is a word or a phase which is replaced with a word, number or symbol.

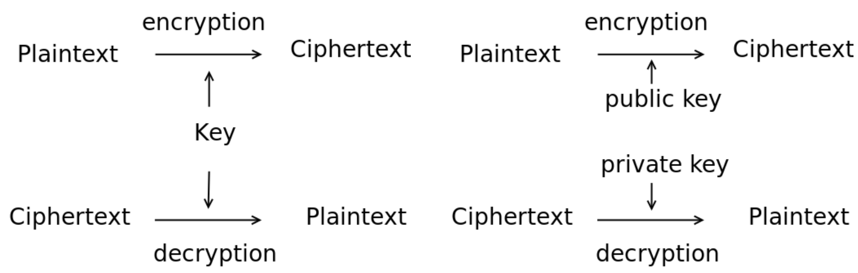
Definition CIPHER
Cipher replaces letters rather than whole word. It should be random gibberish to those not intended to read it.

Theorem KERCHOFF'S PRINCIPLE
"The system should not depend on secrecy, and it should be able to fall into the enemy's hand without disadvantage".

Definition SYMMETRIC CRYPTOGRAPHY
All parties uses same key for encryption and decryption.

1 http://www.barcodesinc.com/articles/cryptography2.htm
2 Applied Cryptography by David Evans

Definition ASYMMETRIC CRYPTOGRAPHY
Different keys for encryption and decryption are used. One key is made public and the other is reserved secret.



(a) Private Key (b) Public Key

Figure 1: Public & Private Key

Definition BLOCK CIPHER
Block cipher can be represented by a bijective function f which accepts as input a block of plaintext of a fixed size, and a key, and outputs a block of ciphertext.[3]

$$f(p, k) = c \tag{1}$$

where
p is a fixed size plaintext
k is the key
c is the cipher text

Definition STREAM CIPHER
Stream ciphers keep some sort of memory, or state, as it processes the plaintext and uses this state as an input to the cipher algorithm. A stream cipher is two functions, f and g.[4]

$$\sigma_{t+1} = f(\sigma_t, p_t, k) \qquad c_t = g(\sigma_t, p_t, k) \tag{2}$$

where
f is a next state function
g is output function
p is a fixed size plaintext
k is the key
c is the cipher text

3 Lecture Notes on Stream Ciphers and RC4 by Rick Wash
4 Lecture Notes on Stream Ciphers and RC4 by Rick Wash

2 DATA ENCRYPTION STANDARD

"In programming, the hard part isn't solving problems, but deciding what problems to solve."

— Paul Graham

Overview

DES is a `Block cipher`. It encypts data in 64 bits blocks. It's a `Symmetric` algorithm. The key length is 56 bits The same algorithm & key is used for both enryption and decryption.

The key is 64 bits but every eight bit is ignored & is used for parity checking.

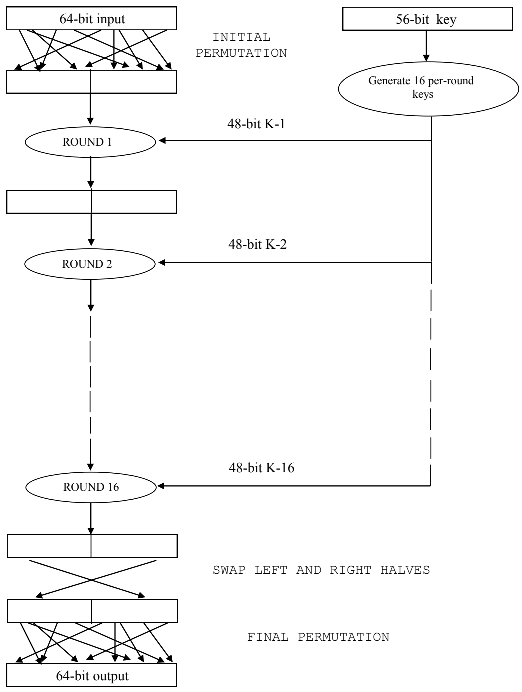


Figure 2: Basic Structure of DES

Description

DES operates on 64 bits block of plaintext. After `initial permutation`, block is broken into two halves of 32 bits. Further 16 `rounds` of identical operations called `function F` combines data with the key. After sixteen rounds, right & left half are joined & a final permutation completes the algorithm.

In each round, the key bits are shifted & then 48 bits are selected from 56 bits of the key. The right half(32 bits) of the data is fed into a `Mangler function`. In this mangler function the data is expanded to 48 bits via an `Expansion Permutation`, combined with 48 bits of a shifted & permuted key via an `XOR` is sent through eight `S-box` producing 32 new bits & permuted again.

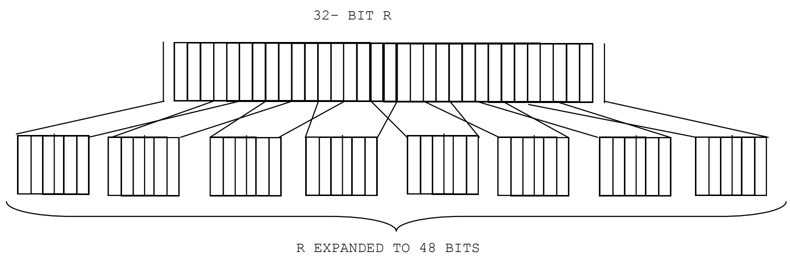


Figure 3: Expansion of Right half to 48 bits

The output of the mangler function is then combined with the left half via another `XOR`. The result of these operations becomes the new right half, the old right half becomes the new left half. These operations are repeated 16 times.

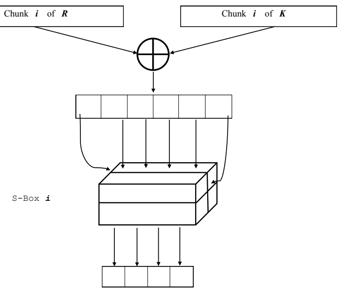


Figure 4: Mangler S-box

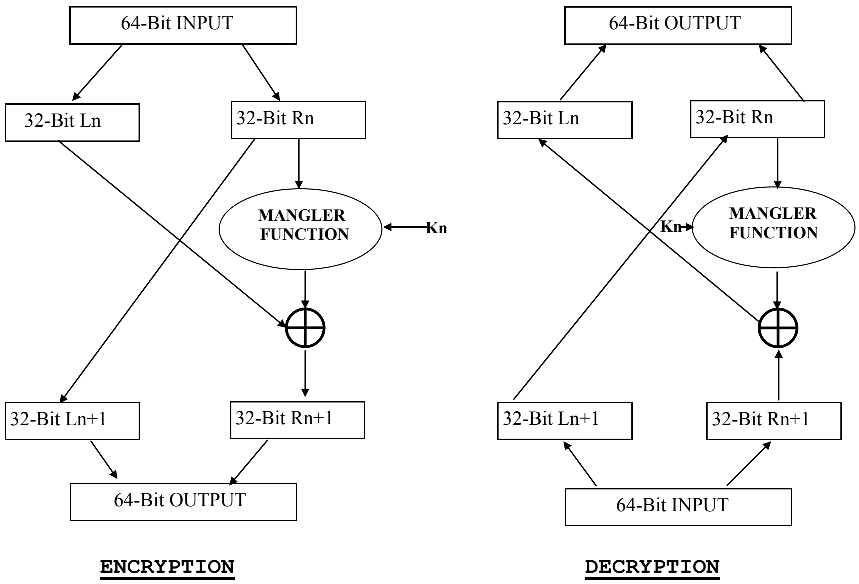


Figure 5: A Complete DES round

If B_i is the result of i^{th} iteration, L_i and R_i are the left and right halves of B_i, K_i is the 48 bit key for round i and function f does all the subsitution and permutation and XORing with the key.[1]

Here

$$L_i = R_{i-1} \qquad R_i = L_{i-1} + f(R_{i-1}, k_i) \tag{3}$$

Initital Permutation
It occurs before round 1. It transposes input block. Initial Permutation and corresponding final permutation doesn't affect DES security.[2]

Expansion Permutation
This operation expands right half of data, R_i, from 32 bits to 48 bits. It has two purposes: It makes the right half the same size as the key for the XOR operation and it provides a longer result can be compressed during substitution.

2.0.1 *Implementation*

```

Encryption

Message in ascii
lastdays
Message in binary
0110110001100001011100110111010001100100011000010111100101110011


Key in ascii
juitworg
Key in binary
0110101001110101011010010111010001110111011011110111001001100111


56 bit binary key
00000000111111111111111101011111000110111010001001011010

Sixteen 48bits keys are
```

1 Applied Cryptography by Bruce Scheneiner
2 Bruce Scheneiner

```
111000001011111001101110011000111000011100110110
111100001011011011110110100101101101001111001010
111101001101011001110110000101001011011101100101
111001101101001101110110111101010101100111000000
101011101101001101110111011010001110111100011011
101011110101001101111011001111110111010000011010
101011110101001111111001111011010101000101100010
100111110101101111011001100001001110101001101110
001111110100100111011011010100010100110101011001
001111110110100110011101110010111011000000011100
000111110010110110011101111000010111011110101100
010111110010110010111101001110000001101010101111
110111111010110010101100110101100101100010110111
110110101010111010101110000001110010101111111001
111110001011111000101110101100111011100101010001
111100011011111010100110111111110000001001001111

Initial Permutation
1111111111001100000110011110011000000000111111110100000110000100

R16 L16 sum
0011101000010101100001100010000100101101001101010101000010111100

Initial Permutation Inverse
1011000101000100101101101100001001111010111000110000100000000110

Cipher in base64
sUS2wnrjCAY=

Decryption

Message in base64
sUS2wnrjCAY=
Message in binary
1011000101000100101101101100001001111010111000110000100000000110


Key in ascii
juitworg
Key in binary
0110101001110101011010010111010001110111011011110111001001100111


56 bit binary key
00000000111111111111111101011111000110111010001001011010
```

```
Sixteen 48bits keys are
111000001011111001101110011000111000011100110110
111100001011011011110110100101101101001111001010
111101001101011001110110000101001011011101100101
111001101101001101110110111110101010110011100000
101011101101001101110111011010001110111100011011
101011110101001101111011001111110111010000011010
101011110101001111111001111011010101000101100010
100111110101101111011001100001001110101001101110
001111110100100111011011010100010100110101011001
001111110110100110011101110010111011000000011100
000111110010110110011101111000010111011110101100
0101111100101100101111010011100000001101010101111
110111111010110010101100110101100101100010110111
110110101010111010101110000000111001010111111001
111110001011111000101110101100111011100101010001
111100011011111010100110111111110000001001001111

Initial Permutation
0011101000010101100001100010000100101101001101010101000010111100

R16 L16 sum
1111111111001100000011001111001100000000001111111101000000110000100

Initial Permutation Inverse
0110110001100001011100110111010001100100011000010111100101110011


Decrypted Message
lastdays
```

3

IDEA

"If you're not failing 90% of the time, then you're probably not working on sufficiently challenging problems."

— Alan Kay.

Overview
IDEA is a block cipher. It operates on 64 bits plaintext blocks. The key is 128 bits long. In IDEA, each primitive operations maps two 16 bits value into one 16 bits value.

IDEA has three primitive operations

1. XOR
2. Addition modulo 2^{16}
3. Multiplicative modulo $2^{16} + 1$

Addition modulo 2^{16}
Addition is done by throwing away carries which is addition is mod 2^{16}.[1]

Multiplicative modulo $2^{16} + 1$
First calculate a 32 bit value by multiplying two 16 bit value and then taking the remainder when divided by $2^{16} + 1$ (65537 in
decimal system). $2^{16} + 1$ is a prime which ensures that every 16 bit
input (0 - 65537) has a multiplicative inverse. Because 0 would not have an inverse, it is expressed as 2^{16}.[2]

Key Generation
IDEA requires 52 keys (6 for each of eight rounds and four more for output transformation. 128 bit key is first divided into

1 Network Security, Private Communication in a Public World, Charlie Kaufman, Radia Perlman, Mike Speciner
2 Network Security, Private Communication in a Public World, Charlie Kaufman, Radia Perlman, Mike Speciner

eight 16 bit subkeys.

For next round, key is rotated 25 bits to the left and again divided into 8 subkeys. The key is rotated again for 25 bits to left for five more rounds until 52 keys are

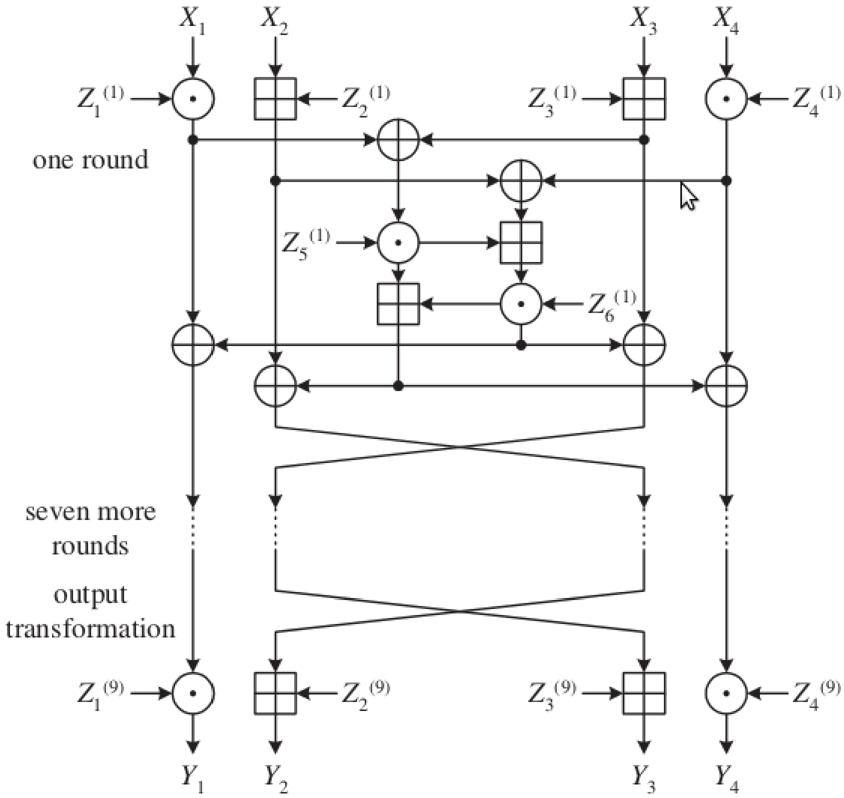


Figure 6: Complete rounds of IDEA

Here

⊕ is XOR
⊞ is modulo multiplicative 2^{16}
⊙ is modulo multiplicative $2^{16} + 1$

In each round, we follow the sequence of events which are as follows:[3]

1. Multiply X1 and the first subkey.
2. Add X2 and the second subkey.
3. Add X3 and the third subkey.

3 Applied Cryptography by Bruce Schneier

4. Multiply X4 and the fourth subkey.
5. XOR the results of steps (1) and (3).
6. XOR the results of steps (2) and (4).
7. Multiply the results of step (5) with the fifth subkey.
8. Add the results of steps (6) and (7).
9. Multiply the results of step (8) with the sixth subkey.
10. Add the results of steps (7) and (9)
11. XOR the results of steps (1) and (9).
12. XOR the results of steps (3) and (9).
13. XOR the results of steps (2) and (10).
14. XOR the results of steps (4) and (10).

Output of round is four subblocks of 12, 13, 14 and 15. Swap two inner blocks i.e. 13 and 14 and that's input to next round. Repeat it for seven more rounds. After eight rounds

1. Multiply X1 with 1^{st} subkey
2. Add X2 and 2^{nd} subkey
3. Add X3 and 3^{rd} subkey
4. Multiply X4 and fourth subkey.

Four subblocks are reattached to form a cipher text.

Generation of Decryption Keys
First four subkeys for decryption

1 / K(x) means multiplicative inverse. -K(x) means additive inverse. x is 1,2...52

1. KD(1) = 1 / K(49)
2. KD(2) = -K(50)
3. KD(3) = -K(51)
4. KD(4) = 1 / K(52)

The remaining keys are

1. Start
2. KD(5) = K(47)
3. KD(6) = K(48)
4. KD(7) = 1 / K(43)
5. KD(8) = -K(45)
6. KD(9) = -K(44)
7. KD(10 = 1 / K(46)
8. Repeat steps (1) eight times

Multiplicative inverse
In order to find multiplicative inverse, we must find greatest common divisor (gcd) of two positive integers.

Euclid Algorithm [4]

Let a, b ε Z. a, b > 0.

1. Let a = 13566 and b = 35742
2. Divide the smaller number into the larger
   b = q * a + r
   Here,
   q = quotient
   r = remainder

   So,
   35742 = 13566 * 2 + 8601
3. Divide the remainder into the previous divisor
   13566 = 1 * 8610 + 4956
4. Continue until remainder is zero

```
8601 = 1 * 4956 + 3654
4956 = 1 * 3654 + 1302
3654 = 1 * 1302 + 1050
```

4 Fall 2006, Chris Christensen

```
1302 = 1 * 1050 + 252
1050 = 4 * 252  + 42
 252 = 6 * 42   + 0
```

5. The gcd of (13566, 35742) is 42.

Relative prime
A pair of positive integers is said to be relative prime if gcd = 1.

```
x * multiplicative_inverse (x) == 1 (modulo 65537)
```

In order to find multiplicative inverse, Extended Euclid algorithm
is used.

Extended Euclidian Algorithm[5]
For any two integers m and n

If m and n are relative prime than
∃ two integers x and y such that
m*x + n*y = 1
- x is the multiplicative inverse of m modulo n
- y is the multiplicative inverse of n modulo m

1. Let m = 65537 and n = user input
   Here (m>n)
2. Set a[0] = m, a[1] = n
3. set x[0] = 1, x[1] = 0, y[0] = 0, y[1] = 1
4. q[k] = floor[a(k-1)/a(k)] for k = 0
5. a[k] = a[k-2] - (a[k-1] * q[k-1]) for k>1
6. x[k] = x[k-2] - (q[k-1] * x[k-1]) for k>1
7. y[k] = y[k-2] - (q[k-1] * y[k-1]),for k>1
   If a[p] is the last non zero a[k] then
   - a[p] = GCD(m,n) = x[p] * m + y[p] * n.
   - x[p] is the multiplicative inverse of m modulo n.
   - y[p] is the multiplicative inverse of n modulo m.

5 Basic Number Theory for RSA Algorithm, Dr. Natarajan Meghanathan, Assistant Professor of Computer Science, Jackson State University

Additive Inverse

```
x + additive_inverse (x) == 0
```

3.1 IMPLEMENTATION

```

Encryption
~~~~~~~~~~

Message in ascii
tusharsh
Message in 64 bit binary
0111010001110101011100110110100001100001011100100111001101101000


Key in ascii
juitworgjuitworg
key in 128 bit binary
0110101001110101011010010111010001110111011011110111001001100111

0110101001110101011010010111010001110111011011110111001001100111


Keys Sequence
0    16   32   48   64   80   96   112
25   41   57   73   89   105  121  9
50   66   82   98   114  2    18   34
75   91   107  123  11   27   43   59
100  116  4    20   36   52   68   84
125  13   29   45   61   77   93   109
22   38   54   70

 Keys
0110101001110101
0110100101110100
0111011101101111
0111001001100111
0110101001110101
0110100101110100
0111011101101111
0111001001100111
1110100011101110
1101111011100100
1100111011010100
```

```
1110101011010010
1110100011101110
1101111011100100
1100111011010100
1110101011010010
1100100110011101
1010100111010101
1010010111010001
1101110110111101
1100100110011101
1010100111010101
1010010111010001
1101110110111101
1010101101001011
1010001110111011
0111101110010011
0011101101010011
1010101101001011
1010001110111011
0111101110010011
0011101101010011
0111011011110111
0010011001110110
1010011101010110
1001011101000111
0111011011110111
0010011001110110
1010011101010110
1001011101000111
1110110101001110
1010110100101110
1000111011101101
1110111001001100
1110110101001110
1010110100101110
1000111011101101
1110111001001100
0101110100011101
1101101111011100
1001100111011010
1001110101011010


****************1 Round**********************

x1 = x1 * k1
```

```
0111101100001100
x2 = x2 + k2
1101110011011100
x3 = x3 + k3
1101100011100001
x4 = x4 * k4
1000101101000110

a1 = x1 xor x3
1010001111101101

a2 = x2 xor x4
0101011110011010

a1 = a1 * k5
1100100100100111

a3 = a1 + a2
0010000011000001

a3 = a3 * k6
1111001011110111

a4 = a1 + a3
1011110000011110

x1 = x1 xor a3
1000100111111011

x3 = x3 xor a3
0010101000010110

x2 = x2 xor a4
0110000011000010

x4 = x4 xor a4
0011011101011000

x2 after swap
0010101000010110
x3 after swap
0110000011000010

****************2 Round**********************

x1 = x1 * k1
```

```
0100000001110110
x2 = x2 + k2
1001110001111101
x3 = x3 + k3
0100100110110000
x4 = x4 * k4
0110101000110001

a1 = x1 xor x3
0000100111000110

a2 = x2 xor x4
1111011001001100

a1 = a1 * k5
0110010000010011

a3 = a1 + a2
0101101001011111

a3 = a3 * k6
1010010100001010

a4 = a1 + a3
0000100100011101

x1 = x1 xor a3
1110010101111100

x3 = x3 xor a3
1110110010111010

x2 = x2 xor a4
1001010101100000

x4 = x4 xor a4
0110001100101100

x2 after swap
1110110010111010
x3 after swap
1001010101100000

****************3 Round*********************

x1 = x1 * k1
```

```
1110100001111100
x2 = x2 + k2
1100101110011110
x3 = x3 + k3
0110010000110100
x4 = x4 * k4
0011011100100001

a1 = x1 xor x3
1000110001001000

a2 = x2 xor x4
1111110010111111

a1 = a1 * k5
0010000110101110

a3 = a1 + a2
0001111001101101

a3 = a3 * k6
0011000110000010

a4 = a1 + a3
0101001100110000

x1 = x1 xor a3
1101100111111110

x3 = x3 xor a3
0101010110110110

x2 = x2 xor a4
1001100010101110

x4 = x4 xor a4
0110010000010001

x2 after swap
0101010110110110
x3 after swap
1001100010101110

****************4 Round**********************

x1 = x1 * k1
```

```
0010000100101100
x2 = x2 + k2
0011001101110011
x3 = x3 + k3
0110001001001011
x4 = x4 * k4
0011100011000011

a1 = x1 xor x3
0100001101100111

a2 = x2 xor x4
0000101110110000

a1 = a1 * k5
0011111001101111

a3 = a1 + a2
0100101000011111

a3 = a3 * k6
0011101110110000

a4 = a1 + a3
0111101000011111

x1 = x1 xor a3
0001101010011100

x3 = x3 xor a3
0101100111111011

x2 = x2 xor a4
0100100101101100

x4 = x4 xor a4
0100001011011100

x2 after swap
0101100111111011
x3 after swap
0100100101101100

****************5 Round**********************

x1 = x1 * k1
```

```
1110110111100111
x2 = x2 + k2
1111110110110110
x3 = x3 + k3
1100010011111111
x4 = x4 * k4
0101000111010110

a1 = x1 xor x3
0010100100011000

a2 = x2 xor x4
1010110001100000

a1 = a1 * k5
1111011010001010

a3 = a1 + a2
1010001011101010

a3 = a3 * k6
1001011010111101

a4 = a1 + a3
1000110101000111

x1 = x1 xor a3
0111101101011010

x3 = x3 xor a3
0101001001000010

x2 = x2 xor a4
0111000011110001

x4 = x4 xor a4
1101110010010001

x2 after swap
0101001001000010
x3 after swap
0111000011110001

****************6 Round**********************

x1 = x1 * k1
```

```
1101011100100100
x2 = x2 + k2
1000110110010101
x3 = x3 + k3
1110011111101000
x4 = x4 * k4
0000111110110011

a1 = x1 xor x3
0011000011001100

a2 = x2 xor x4
1000001000100110

a1 = a1 * k5
0101100010100011

a3 = a1 + a2
1101101011001001

a3 = a3 * k6
1011101101110111

a4 = a1 + a3
0001010000011010

x1 = x1 xor a3
0110110001010011

x3 = x3 xor a3
0101110010011111

x2 = x2 xor a4
1001100110001111

x4 = x4 xor a4
0001101110101001

x2 after swap
0101110010011111
x3 after swap
1001100110001111

****************7 Round**********************

x1 = x1 * k1
```

```
1001001110111111
x2 = x2 + k2
1000001100010101
x3 = x3 + k3
0100000011100101
x4 = x4 * k4
0100101010000111

a1 = x1 xor x3
1101001101011010

a2 = x2 xor x4
1100100110010010

a1 = a1 * k5
1111001110000011

a3 = a1 + a2
1011110100010101

a3 = a3 * k6
1010101011011110

a4 = a1 + a3
1001111001100001

x1 = x1 xor a3
0011100101100001

x3 = x3 xor a3
1110101000111011

x2 = x2 xor a4
0001110101110100

x4 = x4 xor a4
1101010011100110

x2 after swap
1110101000111011
x3 after swap
0001110101110100

****************8 Round*********************

x1 = x1 * k1
```

```
1100110011000101
x2 = x2 + k2
1101100010000111
x3 = x3 + k3
0000101011000010
x4 = x4 * k4
0001111101001111

a1 = x1 xor x3
1100011000000111

a2 = x2 xor x4
1100011111001000

a1 = a1 * k5
1100011111101101

a3 = a1 + a2
1000111110110101

a3 = a3 * k6
0110100111111000

a4 = a1 + a3
0011000111100101

x1 = x1 xor a3
1010010100111101

x3 = x3 xor a3
0110001100111010

x2 = x2 xor a4
1110100101100010

x4 = x4 xor a4
0010111010101010

*************The final Output transformations*******


x1 = x1 * k1
1010010011010000

x2 = x2 + k2
1100010100111110
```

```

x3 = x3 + k3
1111110100010100

x4 = x4 * k4
1000110100010110

Encrypted message
1010010011010000110001010011111011111101000101001000110100010110


Decryption
~~~~~~~~~~

Decryption keys
1110111010100110
0010010000100100
0110011000100110
0011110010001100
1000111011101101
1110111001001100
1001010000011111
0001001010110010
0001000110110100
1001100110100001
1110110101001110
1010110100101110
1101110011101101
0101100010101010
1101100110001010
0001001110110011
1010011101010110
1001011101000111
0000001001111100
1000100100001001
1100010010101101
1110110111000111
1010101101001011
1010001110111011
0011111010001000
1000010001101101
0101110001000101
1100101100111110
1010010111010001
1101110110111101
1010100011101000
```

```
0011011001100011
0010001001000011
1000001010010111
1100100110011101
1010100111010101
0101101000001111
0011000100101100
0010000100011100
1000111101000110
1100111011010100
1110101011010010
1101010000010000
0001011100010010
1000110110011001
0101000101010100
0110101001110101
0110100101110100
1111111110011100
1001011010001100
1000100010010001
0000111110110011

****************1 Round*********************

x1 = x1 * k1
1010010100111101
x2 = x2 + k2
1110100101100010
x3 = x3 + k3
0110001100111010
x4 = x4 * k4
0010111010101010

a1 = x1 xor x3
1100011000000111

a2 = x2 xor x4
1100011111001000

a1 = a1 * k5
1100011111101101

a3 = a1 + a2
1000111110110101
```

```
a3 = a3 ∗ k6
0110100111111000

a4 = a1 + a3
0011000111100101

x1 = x1 xor a3
1100110011000101

x3 = x3 xor a3
0000101011000010

x2 = x2 xor a4
1101100010000111

x4 = x4 xor a4
0001111101001111

x2 after swap
0000101011000010
x3 after swap
1101100010000111

∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗2 Round∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗

x1 = x1 ∗ k1
0011100101100001
x2 = x2 + k2
0001110101110100
x3 = x3 + k3
1110101000111011
x4 = x4 ∗ k4
1101010011100110

a1 = x1 xor x3
1101001101011010

a2 = x2 xor x4
1100100110010010

a1 = a1 ∗ k5
1111001110000011

a3 = a1 + a2
1011110100010101
```

```
a3 = a3 * k6
1010101011011110

a4 = a1 + a3
1001111001100001

x1 = x1 xor a3
1001001110111111

x3 = x3 xor a3
0100000011100101

x2 = x2 xor a4
1000001100010101

x4 = x4 xor a4
0100101010000111

x2 after swap
0100000011100101
x3 after swap
1000001100010101

****************3 Round**********************

x1 = x1 * k1
0110110001010011
x2 = x2 + k2
1001100110001111
x3 = x3 + k3
0101110010011111
x4 = x4 * k4
0001101110101001

a1 = x1 xor x3
0011000011001100

a2 = x2 xor x4
1000001000100110

a1 = a1 * k5
0101100010100011

a3 = a1 + a2
1101101011001001
```

```
a3 = a3 * k6
1011101101110111

a4 = a1 + a3
0001010000011010


x1 = x1 xor a3
1101011100100100

x3 = x3 xor a3
1110011111101000

x2 = x2 xor a4
1000110110010101

x4 = x4 xor a4
0000111110110011

x2 after swap
1110011111101000
x3 after swap
1000110110010101

****************4 Round**********************

x1 = x1 * k1
0111101101011010
x2 = x2 + k2
0111000011110001
x3 = x3 + k3
0101001001000010
x4 = x4 * k4
1101110010010001

a1 = x1 xor x3
0010100100011000

a2 = x2 xor x4
1010110001100000

a1 = a1 * k5
1111011010001010

a3 = a1 + a2
1010001011101010
```

```
a3 = a3 * k6
1001011010111101

a4 = a1 + a3
1000110101000111

x1 = x1 xor a3
1110110111100111

x3 = x3 xor a3
1100010011111111

x2 = x2 xor a4
1111110110110110

x4 = x4 xor a4
0101000111010110

x2 after swap
1100010011111111
x3 after swap
1111110110110110

****************5 Round**********************

x1 = x1 * k1
0001101010011100
x2 = x2 + k2
0100100101101100
x3 = x3 + k3
0101100111111011
x4 = x4 * k4
0100001011011100

a1 = x1 xor x3
0100001101100111

a2 = x2 xor x4
0000101110110000

a1 = a1 * k5
0011111001101111

a3 = a1 + a2
0100101000011111
```

```
a3 = a3 * k6
0011101110110000

a4 = a1 + a3
0111101000011111

x1 = x1 xor a3
0010000100101100

x3 = x3 xor a3
0110001001001011

x2 = x2 xor a4
0011001101110011

x4 = x4 xor a4
0011100011000011

x2 after swap
0110001001001011
x3 after swap
0011001101110011

****************6 Round**********************

x1 = x1 * k1
1101100111111110
x2 = x2 + k2
1001100010101110
x3 = x3 + k3
0101010110110110
x4 = x4 * k4
0110010000010001

a1 = x1 xor x3
1000110001001000

a2 = x2 xor x4
1111110010111111

a1 = a1 * k5
0010000110101110

a3 = a1 + a2
0001111001101101
```

```
a3 = a3 * k6
0011000110000010

a4 = a1 + a3
0101001100110000

x1 = x1 xor a3
1110100001111100

x3 = x3 xor a3
0110010000110100

x2 = x2 xor a4
1100101110011110

x4 = x4 xor a4
0011011100100001

x2 after swap
0110010000110100
x3 after swap
1100101110011110

****************7 Round*********************

x1 = x1 * k1
1110010101111100
x2 = x2 + k2
1001010101100000
x3 = x3 + k3
1110110010111010
x4 = x4 * k4
0110001100101100

a1 = x1 xor x3
0000100111000110

a2 = x2 xor x4
1111011001001100

a1 = a1 * k5
0110010000010011

a3 = a1 + a2
0101101001011111
```

```
a3 = a3 * k6
1010010100001010

a4 = a1 + a3
0000100100011101

x1 = x1 xor a3
0100000001110110

x3 = x3 xor a3
0100100110110000

x2 = x2 xor a4
1001110001111101

x4 = x4 xor a4
0110101000110001

x2 after swap
0100100110110000
x3 after swap
1001110001111101

****************8 Round**********************

x1 = x1 * k1
1000100111111011
x2 = x2 + k2
0110000011000010
x3 = x3 + k3
0010101000010110
x4 = x4 * k4
0011011101011000

a1 = x1 xor x3
1010001111101101

a2 = x2 xor x4
0101011110011010

a1 = a1 * k5
1100100100100111

a3 = a1 + a2
0010000011000001
```

```
a3 = a3 * k6
1111001011110111

a4 = a1 + a3
1011110000011110

x1 = x1 xor a3
0111101100001100

x3 = x3 xor a3
1101100011100001

x2 = x2 xor a4
1101110011011100

x4 = x4 xor a4
1000101101000110

*************The final Output transformations*******


x1 = x1 * k1
0111010001110101

x2 = x2 + k2
0111001101101000

x3 = x3 + k3
0110000101110010

x4 = x4 * k4
0111001101101000

Decrypted message in binary
0111010001110101011100110110100001100001011100100111001101101000


Decrypted message in ascii
tusharsh
```

4 ADVANCED ENCRYPTION STANDARD

"If your password is your name...you deserve to be hacked."

— Anonymous

Overview

Rijndael was selected by NIST (National Institute of Standards and Technology) as AES. AES is a symmetric cipher. Unlike Rijndael in which block length and key length can be specified to any multiple of 32 bits, AES fixes block length of 128 bits and key length to 128 bits, 192 bits or 256 bits.

We however have only worked with key of 128 bits.

Basic Algorithm

The total number of rounds with key of 128 bits is 10. There are four sub-routines (or layers) that are performed in each rounds.

1. Byte Substitution Transformation

Each element of state is non-linearly mapped to the corresponding element in the S-box.

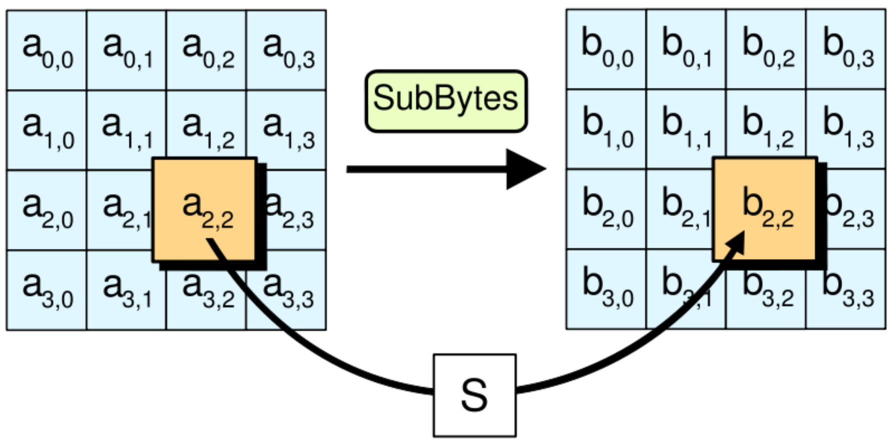


Figure 7: Byte Substitution

2. Shift Rows

Rows are cyclically shifted to the left with offset of 0, 1, 2, 3 for rows of 1, 2, 3 and 4 respectively.

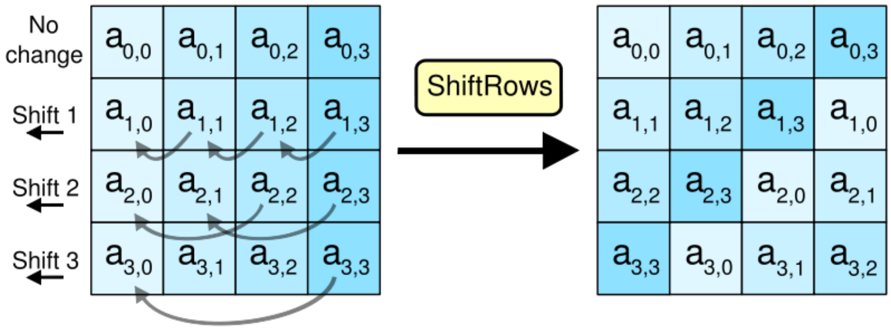


Figure 8: Shift Rows

3. Mix Columns [1]

Each new column (r0, r1, r2, r3) is generated from the old column (a0, a1, a2, a3).

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Figure 9: Mix Columns

Here,
r0 = {2 . a0} + {3 . a1} + {1 . a2} + {1 . a3}
r1 = {1 . a0} + {2 . a1} + {3 . a2} + {1 . a3}
r2 = {1 . a0} + {1 . a1} + {2 . a2} + {1 . a3}
r3 = {3 . a0} + {1 . a1} + {1 . a2} + {2 . a3}

1 Understanding AES Mix-Columns Transformation Calculation by Kit Choy Xintong

Multiplication by 2

1. Let a0 = bf = 10111111
2. set temp = MSB of bf
3. then left shift a0
   a0 = 0111 1110
4. check if temp is 1
5. if yes then
   a0 XOR 0001 1011
6. else
   do nothing

Multiplication by 3

1. Let a0 = bf = 10111111
2. set temp = original number
3. Repeat multiplication by 2
4. a0 XOR temp

Key Generation
Initial Key is described as a state matrix.

C0	**C1**	**C2**	**C3**
a_{00}	a_{01}	a_{02}	a_{03}
b_{10}	b_{11}	b_{12}	b_{13}
c_{20}	c_{21}	c_{22}	c_{23}
d_{30}	d_{31}	d_{32}	d_{33}

The last coloumn is cyclically rotated to move the last block to the top.

C0	**C1**	**C2**	**C3**
a_{00}	a_{01}	a_{02}	d_{33}
b_{10}	b_{11}	b_{12}	a_{03}
c_{20}	c_{21}	c_{22}	b_{13}
d_{30}	d_{31}	d_{32}	c_{23}

The last coloumn is then mapped with the corresponding element of the s-box.

C3		**C3′**
d_{33}	–>	d'_{33}
a_{03}	–>	a'_{03}
b_{13}	–>	b'_{13}
c_{23}	–>	c'_{23}

New column Co is generated by XORing previous Co with the new last column.

Co		**C3′**
a_{00}	$\oplus$	d'_{33}
b_{10}	$\oplus$	a'_{03}
c_{20}	$\oplus$	b'_{13}
d_{30}	$\oplus$	c'_{23}

Similarly all new columns are generated by XORing with previous coloumns.

Decryption

1. Inverse Byte Substitution
It is similar to Byte Substitution but we use inverse s-box instead.

2. Inverse Shift Row
It is similar to Shift Row but the state matrix is cylically rotated to the right instead of left.

3. Inverse Mix Coloumns [2]
Each new column (r0, r1, r2, r3) is generated from the old column (a0, a1, a2, a3).

Here,

r0 = {14 . a0} + {11 . a1} + {13 . a2} + {9 . a3}

r1 = {9 . a0} + {14 . a1} + {11 . a2} + {13 . a3}

r2 = {13 . a0} + {9 . a1} + {14 . a2} + {11 . a3}

r3 = {11 . a0} + {13 . a1} + {9 . a2} + {14 . a3}

2 crypto.stackexchange.com/questions_2569

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Figure 10: Inverse Mix Columns

Multiplication
We can reduce multiplication of 9, 11, 13 and 14 in the multiples of 2's and 3's

1. x * 9 = ((x * 2) * 2) * 2) + x
2. x * 11 = ((((x * 2) * 2) + x) * 2) + x
3. x * 13 = ((((x * 2) + x) * 2) * 2) + x
4. x * 14 = ((((x * 2) + x) * 2) + x) * 2

4.1 IMPLEMENTATION

```

Enter 128 bit message
> life at juit

Enter 128 bit key
> some 128 bit key

128 bit Message is
l |   | j |   |
i | a | u |   |
f | t | i |   |
e |   | t |   |

128 bit Key is
s |   |   |   |
o | 1 | b | k |
m | 2 | i | e |
e | 8 | t | y |

Message in hex
6c  |  20  |  6a  |      |
```

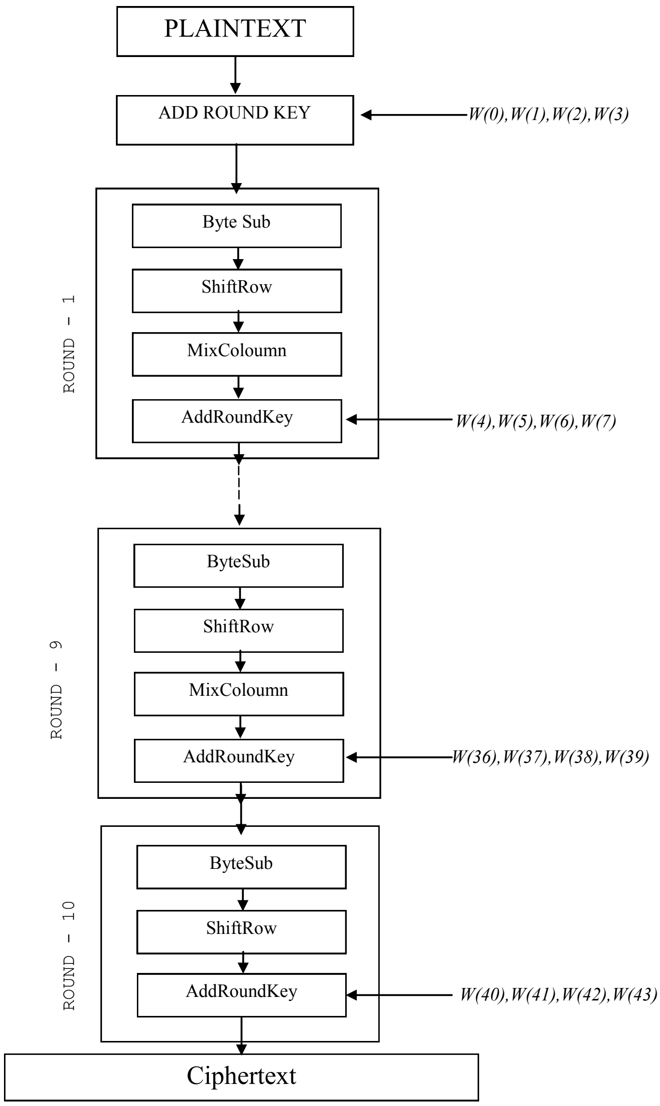


Figure 11: AES

```
69  |  61  |  75  |      |
66  |  74  |  69  |      |
65  |  20  |  74  |      |

key in hex
73  |  20  |  20  |  20  |
6f  |  31  |  62  |  6b  |
6d  |  32  |  69  |  65  |
65  |  38  |  74  |  79  |

Key Generation

key No 1
```

```
0d | 2d | 0d | 2d |
22 | 13 | 71 | 1a |
db | e9 | 80 | e5 |
d2 | ea | 9e | e7 |


key No 2
ad | 80 | 8d | a0 |
fb | e8 | 99 | 83 |
4f | a6 | 26 | c3 |
0a | e0 | 7e | 99 |


key No 3
45 | c5 | 48 | e8 |
d5 | 3d | a4 | 27 |
a1 | 07 | 21 | e2 |
ea | 0a | 74 | ed |


key No 4
81 | 44 | 0c | e4 |
4d | 70 | d4 | f3 |
f4 | f3 | d2 | 30 |
71 | 7b | 0f | e2 |


key No 5
9c | d8 | d4 | 30 |
49 | 39 | ed | 1e |
6c | 9f | 4d | 7d |
18 | 63 | 6c | 8e |


key No 6
ce | 16 | c2 | f2 |
b6 | 8f | 62 | 7c |
75 | ea | a7 | da |
1c | 7f | 13 | 9d |


key No 7
9e | 88 | 4a | b8 |
e1 | 6e | 0c | 70 |
2b | c1 | 66 | bc |
95 | ea | f9 | 64 |
```

```


key No 8
4f | c7 | 8d | 35 |
84 | ea | e6 | 96 |
68 | a9 | cf | 73 |
f9 | 13 | ea | 8e |


key No 9
c4 | 03 | 8e | bb |
0b | e1 | 07 | 91 |
71 | d8 | 17 | 64 |
6f | 7c | 96 | 18 |


key No 10
73 | 70 | fe | 45 |
48 | a9 | ae | 3f |
dc | 04 | 13 | 77 |
85 | f9 | 6f | 77 |


Encryption

Initial Round -ARK
1f  |      |  4a  |  2   |
 6  |  5   |  17  |  6b  |
 b  |  46  |      |  65  |
    |  18  |      |  79  |

 Round 1

Byte Substitution
c0  |  63  |  d6  |  b7  |
6f  |  53  |  f0  |  7f  |
2b  |  5a  |  63  |  4d  |
63  |  ad  |  63  |  b6  |

Shift Rows
c0  |  63  |  d6  |  b7  |
53  |  f0  |  7f  |  6f  |
63  |  4d  |  2b  |  5a  |
b6  |  63  |  ad  |  63  |

Mix Columns
```

```
bb  |  e3  |  b0  |  fd  |
75  |  2c  |  f8  |  e4  |
94  |  ac  |  13  |  c9  |
1c  |  de  |  74  |  31  |

ARK
b6  |  ce  |  bd  |  d   |
57  |  3f  |  89  |  fe  |
4f  |  45  |  93  |  2c  |
ce  |  34  |  ea  |  d6  |

 Round 2

Byte Substitution
4e  |  8b  |  7a  |  70  |
5b  |  75  |  a7  |  bb  |
84  |  6e  |  dc  |  71  |
8b  |  18  |  87  |  f6  |

Shift Rows
4e  |  8b  |  7a  |  70  |
75  |  a7  |  bb  |  5b  |
dc  |  71  |  84  |  6e  |
f6  |  8b  |  18  |  87  |

Mix Columns
29  |  05  |  be  |  e4  |
2d  |  c6  |  98  |  f3  |
99  |  48  |  fa  |  65  |
8c  |  5d  |  81  |  b0  |

ARK
84  |  85  |  33  |  44  |
d6  |  2e  |   1  |  7   |
d6  |  ee  |  dc  |  a6  |
86  |  bd  |  ff  |  29  |

 Round 3

Byte Substitution
5f  |  97  |  c3  |  1b  |
f6  |  31  |  7c  |  51  |
f6  |  28  |  86  |  24  |
44  |  7a  |  16  |  a5  |

Shift Rows
```

```
5f  |  97  |  c3  |  1b  |
31  |  7c  |  51  |  f6  |
86  |  24  |  f6  |  28  |
a5  |  44  |  7a  |  16  |

Mix Columns
ce  |  d1  |  e2  |  09  |
09  |  47  |  1a  |  82  |
8d  |  6f  |  eb  |  87  |
07  |  72  |  0d  |  df  |

ARK
8b  |  14  |  aa  |  e1  |
dc  |  7a  |  be  |  a5  |
2c  |  68  |  ca  |  65  |
ed  |  78  |  79  |  32  |

 Round 4

Byte Substitution
3d  |  fa  |  ac  |  f8  |
86  |  da  |  ae  |  06  |
71  |  45  |  74  |  4d  |
55  |  bc  |  b6  |  23  |

Shift Rows
3d  |  fa  |  ac  |  f8  |
da  |  ae  |  06  |  86  |
74  |  4d  |  71  |  45  |
23  |  55  |  bc  |  b6  |

Mix Columns
58  |  1e  |  84  |  89  |
2d  |  3f  |  8f  |  96  |
6a  |  31  |  97  |  35  |
af  |  5c  |  fb  |  a7  |

ARK
d9  |  5a  |  88  |  6d  |
6   |  4f  |  5b  |  65  |
9e  |  c2  |  45  |   5  |
de  |  27  |  f4  |  45  |

 Round 5

Byte Substitution
```

```
35  |  be  |  c4  |  3c  |
d0  |  84  |  39  |  4d  |
0b  |  25  |  6e  |  6b  |
1d  |  cc  |  bf  |  6e  |

Shift Rows
35  |  be  |  c4  |  3c  |
84  |  39  |  4d  |  d0  |
6e  |  6b  |  0b  |  25  |
6e  |  1d  |  cc  |  bf  |

Mix Columns
fd  |  5a  |  83  |  89  |
fa  |  6c  |  8f  |  57  |
df  |  76  |  d0  |  7c  |
69  |  b1  |  92  |  d4  |

ARK
61  |  82  |  57  |  b9  |
b3  |  55  |  62  |  49  |
b3  |  e9  |  9d  |   1  |
71  |  d2  |  fe  |  5a  |

 Round 6

Byte Substitution
ef  |  13  |  5b  |  56  |
6d  |  fc  |  aa  |  3b  |
6d  |  1e  |  5e  |  7c  |
a3  |  b5  |  bb  |  be  |

Shift Rows
ef  |  13  |  5b  |  56  |
fc  |  aa  |  3b  |  6d  |
5e  |  7c  |  6d  |  1e  |
be  |  a3  |  b5  |  bb  |

Mix Columns
3a  |  1c  |  23  |  be  |
50  |  7b  |  2f  |  15  |
76  |  bf  |  7e  |  d1  |
ef  |  be  |  ca  |  e4  |

ARK
f4  |   a  |  e1  |  4c  |
e6  |  f4  |  4d  |  69  |
```

```
 3  |  55  |  d9  |   b  |
f3  |  c1  |  d9  |  79  |

 Round 7

Byte Substitution
bf  |  67  |  f8  |  29  |
8e  |  bf  |  e3  |  f9  |
7b  |  fc  |  35  |  2b  |
0d  |  78  |  35  |  b6  |

Shift Rows
bf  |  67  |  f8  |  29  |
bf  |  e3  |  f9  |  8e  |
35  |  2b  |  7b  |  fc  |
b6  |  0d  |  78  |  35  |

Mix Columns
3c  |  d6  |  f8  |  12  |
33  |  ca  |  e4  |  04  |
ab  |  c5  |  7f  |  1b  |
27  |  7b  |  61  |  63  |

ARK
a2  |  5e  |  b2  |  aa  |
d2  |  a4  |  e8  |  74  |
8   |   4  |  19  |  a7  |
b2  |  91  |  98  |   7  |

 Round 8

Byte Substitution
3a  |  58  |  37  |  ac  |
b5  |  49  |  9b  |  92  |
cd  |  f2  |  d4  |  5c  |
37  |  81  |  46  |  c5  |

Shift Rows
3a  |  58  |  37  |  ac  |
49  |  9b  |  92  |  b5  |
d4  |  5c  |  cd  |  f2  |
c5  |  37  |  81  |  46  |

Mix Columns
be  |  6d  |  8f  |  33  |
0a  |  a6  |  c5  |  96  |
```

```
94  |  22  |  bc  |  2c  |
42  |  41  |  1f  |  24  |

ARK
f1  |  aa  |   2  |   6  |
8e  |  4c  |  23  |      |
fc  |  8b  |  73  |  5f  |
bb  |  52  |  f5  |  aa  |

 Round 9

Byte Substitution
a1  |  ac  |  77  |  6f  |
19  |  29  |  26  |  63  |
b0  |  3d  |  8f  |  cf  |
ea  |  00  |  e6  |  ac  |

Shift Rows
a1  |  ac  |  77  |  6f  |
29  |  26  |  63  |  19  |
8f  |  cf  |  b0  |  3d  |
ac  |  ea  |  00  |  e6  |

Mix Columns
01  |  0c  |  fb  |  2e  |
d5  |  40  |  7a  |  fc  |
62  |  2a  |  6f  |  3d  |
1d  |  c9  |  4a  |  42  |

ARK
c5  |   f  |  75  |  95  |
de  |  a1  |  7d  |  6d  |
13  |  f2  |  78  |  59  |
72  |  b5  |  dc  |  5a  |

 Round  10

Byte Substitution
a6  |  76  |  9d  |  2a  |
1d  |  32  |  ff  |  3c  |
7d  |  89  |  bc  |  cb  |
40  |  d5  |  86  |  be  |

Shift Rows
a6  |  76  |  9d  |  2a  |
32  |  ff  |  3c  |  1d  |
```

```
bc  |  cb  |  7d  |  89  |
be  |  40  |  d5  |  86  |

ARK
d5  |   6  |  63  |  6f  |
7a  |  56  |  92  |  22  |
6   |  cf  |  6e  |  fe  |
3b  |  b9  |  ba  |  f1  |


Decryption

Initial Round

ARK
a6  |  76  |  9d  |  2a  |
32  |  ff  |  3c  |  1d  |
bc  |  cb  |  7d  |  89  |
be  |  4   |  d5  |  86  |

Inverse Shift Rows
a6  |  76  |  9d  |  2a  |
1d  |  32  |  ff  |  3c  |
7d  |  89  |  bc  |  cb  |
4   |  d5  |  86  |  be  |

Inverse Byte Substitution
c5  |  0f  |  75  |  95  |
de  |  a1  |  7d  |  6d  |
13  |  f2  |  78  |  59  |
72  |  b5  |  dc  |  5a  |

Round No 1

ARK
 1  |   c  |  fb  |  2e  |
d5  |  4   |  7a  |  fc  |
62  |  2a  |  6f  |  3d  |
1d  |  c9  |  4a  |  42  |

Inverse Mix Columns
a1  |  ac  |  77  |  6f  |
29  |  26  |  63  |  19  |
8f  |  cf  |  b0  |  3d  |
ac  |  ea  |  00  |  e6  |
```

```
Inverse Shift Rows
a1  |  ac  |  77  |  6f  |
19  |  29  |  26  |  63  |
b0  |  3d  |  8f  |  cf  |
ea  |  00  |  e6  |  ac  |

Inverse Byte Substitution
f1  |  aa  |  02  |  06  |
8e  |  4c  |  23  |  00  |
fc  |  8b  |  73  |  5f  |
bb  |  52  |  f5  |  aa  |

Round No 2

ARK
be  |  6d  |  8f  |  33  |
 a  |  a6  |  c5  |  96  |
94  |  22  |  bc  |  2c  |
42  |  41  |  1f  |  24  |

Inverse Mix Columns
3a  |  58  |  37  |  ac  |
49  |  9b  |  92  |  b5  |
d4  |  5c  |  cd  |  f2  |
c5  |  37  |  81  |  46  |

Inverse Shift Rows
3a  |  58  |  37  |  ac  |
b5  |  49  |  9b  |  92  |
cd  |  f2  |  d4  |  5c  |
37  |  81  |  46  |  c5  |

Inverse Byte Substitution
a2  |  5e  |  b2  |  aa  |
d2  |  a4  |  e8  |  74  |
80  |  04  |  19  |  a7  |
b2  |  91  |  98  |  07  |

Round No 3

ARK
3c  |  d6  |  f8  |  12  |
33  |  ca  |  e4  |   4  |
ab  |  c5  |  7f  |  1b  |
27  |  7b  |  61  |  63  |
```

```
Inverse Mix Columns
bf  |  67  |  f8  |  29  |
bf  |  e3  |  f9  |  8e  |
35  |  2b  |  7b  |  fc  |
b6  |  0d  |  78  |  35  |

Inverse Shift Rows
bf  |  67  |  f8  |  29  |
8e  |  bf  |  e3  |  f9  |
7b  |  fc  |  35  |  2b  |
0d  |  78  |  35  |  b6  |

Inverse Byte Substitution
f4  |  0a  |  e1  |  4c  |
e6  |  f4  |  4d  |  69  |
03  |  55  |  d9  |  0b  |
f3  |  c1  |  d9  |  79  |

Round No 4

ARK
3a  |  1c  |  23  |  be  |
5   |  7b  |  2f  |  15  |
76  |  bf  |  7e  |  d1  |
ef  |  be  |  ca  |  e4  |

Inverse Mix Columns
ef  |  13  |  5b  |  56  |
fc  |  aa  |  3b  |  6d  |
5e  |  7c  |  6d  |  1e  |
be  |  a3  |  b5  |  bb  |

Inverse Shift Rows
ef  |  13  |  5b  |  56  |
6d  |  fc  |  aa  |  3b  |
6d  |  1e  |  5e  |  7c  |
a3  |  b5  |  bb  |  be  |

Inverse Byte Substitution
61  |  82  |  57  |  b9  |
b3  |  55  |  62  |  49  |
b3  |  e9  |  9d  |  01  |
71  |  d2  |  fe  |  5a  |

Round No 5
```

```
ARK
fd  |  5a  |  83  |  89  |
fa  |  6c  |  8f  |  57  |
df  |  76  |  d   |  7c  |
69  |  b1  |  92  |  d4  |

Inverse Mix Columns
35  |  be  |  c4  |  3c  |
84  |  39  |  4d  |  d0  |
6e  |  6b  |  0b  |  25  |
6e  |  1d  |  cc  |  bf  |

Inverse Shift Rows
35  |  be  |  c4  |  3c  |
d0  |  84  |  39  |  4d  |
0b  |  25  |  6e  |  6b  |
1d  |  cc  |  bf  |  6e  |

Inverse Byte Substitution
d9  |  5a  |  88  |  6d  |
60  |  4f  |  5b  |  65  |
9e  |  c2  |  45  |  05  |
de  |  27  |  f4  |  45  |

Round No 6

ARK
58  |  1e  |  84  |  89  |
2d  |  3f  |  8f  |  96  |
6a  |  31  |  97  |  35  |
af  |  5c  |  fb  |  a7  |

Inverse Mix Columns
3d  |  fa  |  ac  |  f8  |
da  |  ae  |  06  |  86  |
74  |  4d  |  71  |  45  |
23  |  55  |  bc  |  b6  |

Inverse Shift Rows
3d  |  fa  |  ac  |  f8  |
86  |  da  |  ae  |  06  |
71  |  45  |  74  |  4d  |
55  |  bc  |  b6  |  23  |

Inverse Byte Substitution
8b  |  14  |  aa  |  e1  |
```

```
dc  |  7a  |  be  |  a5  |
2c  |  68  |  ca  |  65  |
ed  |  78  |  79  |  32  |

Round No 7

ARK
ce  |  d1  |  e2  |   9  |
 9  |  47  |  1a  |  82  |
8d  |  6f  |  eb  |  87  |
 7  |  72  |   d  |  df  |

Inverse Mix Columns
5f  |  97  |  c3  |  1b  |
31  |  7c  |  51  |  f6  |
86  |  24  |  f6  |  28  |
a5  |  44  |  7a  |  16  |

Inverse Shift Rows
5f  |  97  |  c3  |  1b  |
f6  |  31  |  7c  |  51  |
f6  |  28  |  86  |  24  |
44  |  7a  |  16  |  a5  |

Inverse Byte Substitution
84  |  85  |  33  |  44  |
d6  |  2e  |  01  |  70  |
d6  |  ee  |  dc  |  a6  |
86  |  bd  |  ff  |  29  |

Round No 8

ARK
29  |   5  |  be  |  e4  |
2d  |  c6  |  98  |  f3  |
99  |  48  |  fa  |  65  |
8c  |  5d  |  81  |  b   |

Inverse Mix Columns
4e  |  8b  |  7a  |  70  |
75  |  a7  |  bb  |  5b  |
dc  |  71  |  84  |  6e  |
f6  |  8b  |  18  |  87  |

Inverse Shift Rows
4e  |  8b  |  7a  |  70  |
```

```
5b  |  75  |  a7  |  bb  |
84  |  6e  |  dc  |  71  |
8b  |  18  |  87  |  f6  |

Inverse Byte Substitution
b6  |  ce  |  bd  |  d0  |
57  |  3f  |  89  |  fe  |
4f  |  45  |  93  |  2c  |
ce  |  34  |  ea  |  d6  |

Round No 9

ARK
bb  |  e3  |  b   |  fd  |
75  |  2c  |  f8  |  e4  |
94  |  ac  |  13  |  c9  |
1c  |  de  |  74  |  31  |

Inverse Mix Columns
c0  |  63  |  d6  |  b7  |
53  |  f0  |  7f  |  6f  |
63  |  4d  |  2b  |  5a  |
b6  |  63  |  ad  |  63  |

Inverse Shift Rows
c0  |  63  |  d6  |  b7  |
6f  |  53  |  f0  |  7f  |
2b  |  5a  |  63  |  4d  |
63  |  ad  |  63  |  b6  |

Inverse Byte Substitution
1f  |  00  |  4a  |  20  |
06  |  50  |  17  |  6b  |
0b  |  46  |  00  |  65  |
00  |  18  |  00  |  79  |

Round 10 - ARK
6c  |  2   |  6a  |      |
69  |  61  |  75  |      |
66  |  74  |  69  |      |
65  |  2   |  74  |      |

Decrypted Message
l |   | j |   |
i | a | u |   |
f | t | i |   |
```

```
e |   | t |   |

life at juit
```

5 RC4

"Foolproof systems don't take into account the ingenuity of fools."

— Gene Brown

Overview

RC4 is a binary additive stream cipher. It is used in SSL (also known as TLS), WEP and IEEE 802.11 wireless networking security standard[1]

Basic Algorithm [2]

Generate a state table of size 2^n words. Identity permutation is used to initialize state table (also called s-box).

```
For i = 0 to 2^n - 1
  S[i] = i
```

Scramble the state table using the key as seed.

```
j = 0
for i = 0 to 2^n - 1
  j = (j + S[i] + key[i mod l]) mod
  Swap (S[i], S[j])
```

For the keystream generator, n bit word as keystream is produced

```
Initialization
    i = 0
    j = 0

Generating loop:
    i = (i + 1) mod 256
    j = (j + 1) mod 256
```

1 http://en.wikipedia.org/wiki/RC4

2 Evaluation of the RC4 Algorithm for Data Encryption by Allam Mousa and Ahmad Hamad

```
    Swap (S[i], S[j])
    Output z = S[S[i] + S[j]]
```

The Output is XORed with plaintext to produce ciphertext. The cipher text is fed into the same function for decryption.

5.1 Implementation

```
Enter plaintext > This is a secret message

Enter key between 1 and 256 bytes > waknaghat

Initialized State Matrix
[0, 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, 128, 129, 130, 131, 132, 133,
    134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144,
    145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
    156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166,
    167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177,
    178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188,
    189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199,
    200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210,
    211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221,
    222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232,
    233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243,
    244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254,
    255]

Scrambled State Matrix
[49, 217, 114, 36, 134, 17, 246, 122, 41, 90, 197, 188, 67,
    109, 152, 0, 57, 136, 42, 27, 180, 38, 123, 69, 222,
    243, 204, 126, 33, 23, 140, 181, 13, 120, 92, 147, 187,
    88, 24, 103, 201, 93, 142, 158, 62, 226, 113, 59, 12,
    101, 14, 111, 68, 37, 9, 106, 73, 196, 84, 51, 32, 154,
    191, 189, 220, 173, 153, 155, 253, 151, 39, 175, 205,
```

```
    104, 119, 157, 25, 156, 203, 22, 52, 182, 186, 202, 215,
     233, 97, 108, 223, 55, 34, 143, 159, 237, 118, 83, 145,
     64, 71, 7, 20, 66, 72, 21, 6, 199, 141, 46, 192, 35,
    161, 133, 96, 166, 117, 251, 127, 30, 26, 236, 238, 210,
     81, 209, 61, 235, 148, 102, 218, 163, 76, 130, 230, 29,
     250, 248, 40, 48, 206, 85, 170, 146, 193, 224, 247, 1,
    124, 128, 171, 82, 184, 176, 179, 229, 174, 137, 242,
    214, 167, 194, 195, 172, 241, 245, 138, 45, 8, 232, 135,
     31, 19, 95, 216, 254, 77, 255, 225, 5, 240, 212, 208,
    169, 150, 10, 185, 94, 112, 219, 89, 87, 105, 78, 80,
    221, 228, 116, 60, 125, 144, 129, 162, 107, 4, 110, 100,
     190, 239, 207, 160, 0, 43, 58, 53, 70, 98, 56, 132,
    227, 178, 183, 165, 28, 121, 65, 0, 249, 211, 50, 168,
    234, 252, 139, 44, 99, 11, 54, 177, 16, 86, 200, 244,
    63, 47, 131, 75, 115, 149, 231, 18, 74, 164, 198, 3, 2,
    213, 79]

Cipher is  [13, 104, 118, 57, 90, 8, 162, 197, 249, 135,
    235, 228, 16, 162, 214, 164, 27, 9, 249, 56, 145, 255,
    255, 10]

Decryption

Initialized State Matrix
[0, 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, 128, 129, 130, 131, 132, 133,
    134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144,
    145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
    156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166,
    167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177,
    178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188,
    189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199,
    200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210,
    211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221,
    222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232,
    233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243,
    244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254,
    255]
```

```
Scrambled State Matrix
[49, 217, 114, 36, 134, 17, 246, 122, 41, 90, 197, 188, 67,
    109, 152, 0, 57, 136, 42, 27, 180, 38, 123, 69, 222,
    243, 204, 126, 33, 23, 140, 181, 13, 120, 92, 147, 187,
    88, 24, 103, 201, 93, 142, 158, 62, 226, 113, 59, 12,
    101, 14, 111, 68, 37, 9, 106, 73, 196, 84, 51, 32, 154,
    191, 189, 220, 173, 153, 155, 253, 151, 39, 175, 205,
    104, 119, 157, 25, 156, 203, 22, 52, 182, 186, 202, 215,
     233, 97, 108, 223, 55, 34, 143, 159, 237, 118, 83, 145,
     64, 71, 7, 20, 66, 72, 21, 6, 199, 141, 46, 192, 35,
    161, 133, 96, 166, 117, 251, 127, 30, 26, 236, 238, 210,
     81, 209, 61, 235, 148, 102, 218, 163, 76, 130, 230, 29,
     250, 248, 40, 48, 206, 85, 170, 146, 193, 224, 247, 1,
    124, 128, 171, 82, 184, 176, 179, 229, 174, 137, 242,
    214, 167, 194, 195, 172, 241, 245, 138, 45, 8, 232, 135,
     31, 19, 95, 216, 254, 77, 255, 225, 5, 240, 212, 208,
    169, 150, 10, 185, 94, 112, 219, 89, 87, 105, 78, 80,
    221, 228, 116, 60, 125, 144, 129, 162, 107, 4, 110, 100,
     190, 239, 207, 160, 0, 43, 58, 53, 70, 98, 56, 132,
    227, 178, 183, 165, 28, 121, 65, 0, 249, 211, 50, 168,
    234, 252, 139, 44, 99, 11, 54, 177, 16, 86, 200, 244,
    63, 47, 131, 75, 115, 149, 231, 18, 74, 164, 198, 3, 2,
    213, 79]

Message is
T h i s   i s   a   s e c r e t   m e s s a g e
```

APPENDIX A

We have uploaded our programs to Paste Bin to the following link. Since **IDEA** is patented, we preferred not to upload it due to copyright infringement.

Algorithm	URL1	URL2
DES	http://pastebin.com/kWifSi7A	
AES	http://pastebin.com/fz5CbpyR	http://pastebin.com/toxQtD4t
RC4	http://pastebin.com/wj3ep1Pf	

BIBLIOGRAPHY

[1] Charlie Kauffman, Radia Perlman, Mike Speciner, *Network Security, private communication in a public world*. Pearson Education, 2nd Edition, 2005.

[2] Bruce Schneier, *Applied Cryptography*. 2nd Edition.

[3] Wade Trappe, Lawrence C. Washington, *Introduction to Cryptography with Coding Theory*. Pearson Education, 2nd Edition.

[4] Cetin Kaya Koc, *Cryptographic Engineering*. Springer.

[5] Orlin Grabbe, *The DES Algorithm Illustrated*.

[6] Wikipedia, *International Data Encryption Algorithm*.

[7] How-Shen Chang, *International Data Encryption Algorithm*.

[8] Dr. Natarajan Meghanathan, *Basic Number Theory for RSA*.

[9] NICK HOFFMAN, *A SIMPLIFIED IDEA ALGORITHM*.

[10] Joan Daemen, Vincent Rijmen, *AES Proposal: Rijndael*.

[11] Kit Choy Xintong, *Understanding AES Mix-Columns Transformation Calculation*.

[12] Crypto.stackexchange.com, *How does one implement the Inverse of AES MixColumns*

[13] Allam Mousa and Ahmad Hamad , *Evaluation of the RC4 Algorithm for Data Encryption*

[14] Rick Wah, *Lecture Notes on Stream Cipher and RC4*