Chapter – V

Modified Rotational Encoder (MRE)

Chapter-V

Rotational Encoding

5 Introduction

Rotational Encoding is another kind of encoding. In this technique a text message can be considered as binary string of block length n. The binary string can be considered as bits in a ring. The bits can be rotated right or left. In each rotation there will be one bit shift to the right or left and the least significant (lsb) will occupy the place of most significant bit (msb) or msb will occupy the place of lsb. A string will generate (n-1) different possible string in right or left rotation and in n^{th} rotation the original string will appear for n bit string. It can be used for encryption of message. The message is converted into a binary level and each block with n bit is allowed to rotate through the rotational encoding and is transformed into cipher text. This cipher text may be used for transmission to other places through the open line via internet. Then at the receiving end the cipher text is allowed to decode through its decoding process which is the same as the encoding process. Both encoding and decoding are described here in detail. The principle is tested and verified using a microprocessor based system.

5.1 Principle of Rotational Encoding

A text message consists of blocks of binary string is first converted into binary level. Then each block of binary string of length (n) is given m number of rotation which less than n making a different set of string. This generated string can be used for cipher text. So there is $(n-1)^{th}$ possible encoded string to be generated for n bit block length. To decode the encoded string, (n-m) number of rotation has to be applied at the time of decoding. The principle of this encoding is explained in detail as in the following.

Considering a very simplest case with one byte string is shown in fig 5.1. It is allowed to rotate anticlockwise by one bit i.e. shift right of all the bits by one bit and the lsb will occupy the msb position.

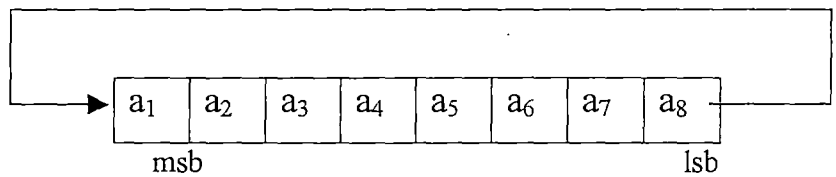

Fig 5.1: 8-bit string under Rotational Encoding

The string under consideration: $a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8$

where a_1 is the most significant bit (msb) and a_8 the least significant bit (lsb). Since the bits considered in the string are 8, there will be rotation required to appear the original string. The rotational encoding is explained with the help of the table 5.1. Here after the first rotation by one bit, lsb (a_8) has taken the msb position and all other bits are shifted right by one bit. So 8 such rotations are required to get back the original string for an 8 bit string and one of the seven intermediate strings can be used as encoded string.

Table 5.1: Intermediately generated string under Rotational Encoding

Original String	$a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8$
String after 1st rotation	$a_8\ a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7$
String after 2nd rotation	$a_7\ a_8\ a_1\ a_2\ a_3\ a_4\ a_5\ a_6$
String after 3rd rotation	$a_6\ a_7\ a_8\ a_1\ a_2\ a_3\ a_4\ a_5$
String after 4th rotation	$a_5\ a_6\ a_7\ a_8\ a_1\ a_2\ a_3\ a_4$
String after 5th rotation	$a_4\ a_5\ a_6\ a_7\ a_8\ a_1\ a_2\ a_3$
String after 6th rotation	$a_3\ a_4\ a_5\ a_6\ a_7\ a_8\ a_1\ a_2$
String after 7th rotation	$a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8\ a_1$
String after 8th rotation	$a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8$

If the string generated after 2nd rotation is used as encoded string, then (8 –2) = 6 more rotations are to applied on the encoded string to get back the original string.

The principle of encoding can be extended to n byte string. The number of rotation required to get back the original string for n byte string (m) = n x 8.

Where n is the number of bytes in the string.

The total number of intermediately generated string, (k) = (n x 8 –1)

For a single byte string, n = 1, k = 7.

Considering that after i-th rotation, the generated string is used as encoded string. Then the number of rotations (x) to be applied on the encoded string at the time of decoding = n x 8 – i.

For n = 1 and i = 2 , then l = 6.

To explain the multi-byte string, a 4 byte string has been considered. The Least Significant Byte (LSB) and the Most Significant Byte (MSB) are shown in fig 5.2. The 4 byte string, is stored byte-wise in the memory as shown in fig 5.3. The LSB is stored at lower location of the memory and MSB part at higher location. When the string is under rotational encoding, it will be assumed to be an endless ring. Each rotation anticlockwise moves the least significant bit to the most significant bit position and all other bits are shifted one bit to the right.

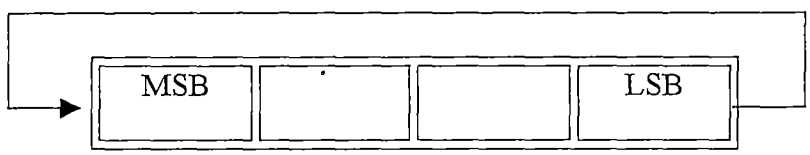

Fig 5.2 : Multi-byte String under Rotational Encoding

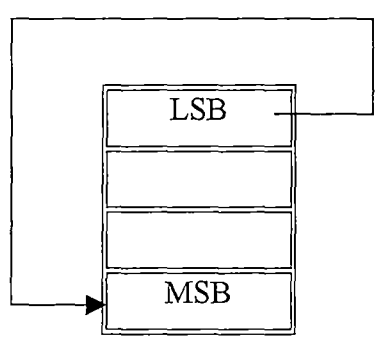

Fig 5.3: Multi-byte String stored in memory under Rotational Encoding

When a large number of bytes are taken into consideration in the string, the rotational encoding will not be very effective. On 8^{th} rotation, the LS byte will go to the MS byte position and all other bytes will be moved to the right. The characters in the string will appear again in the shifted condition and LS byte character will come to the MS byte position. On 16^{th} rotation the same thing will happen. So, after 8 and its multiple rotations the part of the message will reappear as shifted cut and paste condition. This is the disadvantage with the rotational encoding.

5.2 Modification on Rotational Encoding

A modification on rotational encoding is suggested with a view to eliminate the disadvantage associated with the rotational encoding.

Before applying the rotational encoding, a particular bit (say, lsb) of each byte of the string under consideration is complemented. This additional feature is very effective and will eliminate the disadvantage of re-appearing the bytes after 8 and its multiple rotations. This will also be very effective for any number of bytes. The encoding with large number of bytes with a particular bit inverted will be more effective. The complexity will be high with large number of bits in the string.

5.3 Process of Modified Rotational Encoding

It requires two following steps to encode the string for rotational encoding.

1. Inversion of a bit in each byte of the string.
2. Rotational encoding: The number of rotation will be applied to the string under consideration, as suggested in section 5.2.

5.4 Process of Modified Rotational Decoding

Here also two following steps in succession are required to decode the encoded string.

1. Rotational Decoding: The rest number of rotation will be applied to the encoded string under consideration to complete the number of rotations required to get back the original string.
2. Inversion of the bit in each byte of the string.

5.5 Illustration of Modified Rotational Encoding with Example

To illustrate Modified Rotational Encoder (MRE) an example is given, the string length is considered as 8bit. The binary string assumed is 01100011. The string is shown in the first row of table 5.2. After the lsb inversion the string (01100010) is shown in the second row of same table. Then say 3 rotations are applied on it and the generated string (01001100) is the encoded string.

Table 5.2: Rotational Encoding

0	1	1	0	0	0	1	1	Assumed String
0	1	1	0	0	0	1	0	String after lsb inversion
0	1	1	0	0	0	1	0	String under rotation
0	0	1	1	0	0	0	1	1^{st} rotation
1	0	0	1	1	0	0	0	2^{nd} rotation
0	1	0	0	1	1	0	0	3^{rd} rotation

In order to decode the encoded string, (8-3) number of rotation will be applied first, then the lsb bit of the string will be inverted. This is shown in table 5.3 and the original string has been generated as shown in the last row of the table 5.3.

Table 5.3: Rotational Decoding

0	1	0	0	1	1	0	0	Encoded String
0	0	1	0	0	1	1	0	1^{st} rotation
0	0	0	1	0	0	1	1	2^{nd} rotation
1	0	0	0	1	0	0	1	3^{rd} rotation
1	1	0	0	0	1	0	0	4^{th} rotation
0	1	1	0	0	0	1	0	5^{th} rotation
0	1	1	0	0	0	1	1	Original String (After lsb inversion)

The rotational transformation is a symmetric in nature. But the rotational transformation along with the inversion of least significant bit is not a symmetric one, rather asymmetric. Since the same transformation applying successively on the generated string will not produce the original string. It is a kind of cascading the two schemes.

5.6 Realization of MRE using Microprocessor Based System

A microprocessor based system has been used for realizing the Modified Rotational Encoder. To realize the encoder, three following routines are required. The main routine is calling the routines.

a) Routine **lsbinv** – This routine will invert the least significant bit of each byte in the string under consideration.

b) Routine **rot** – This routine will rotate the string of n bytes by one bit in anticlockwise.

c) Routine **'store'** – This routine will store the string as well as the intermediate strings generated.

The routines are given in details in section 5.6.1 to 5.6.6.

5.6.1 Algorithm of routine 'lsbinv'

This routine has used HL pair as memory pointer and C register as counter, representing the number of bytes of which the ls bit will be inverted. The fig 5.4 shows n byte string stored in memory, the Least Significant Byte (LSB) being the first byte, stored in the lower memory and Most Significant Byte (MSB) being the n-th byte, in higher memory address. The least significant bits (lsb) and msb are shown in the figure 5.4. The

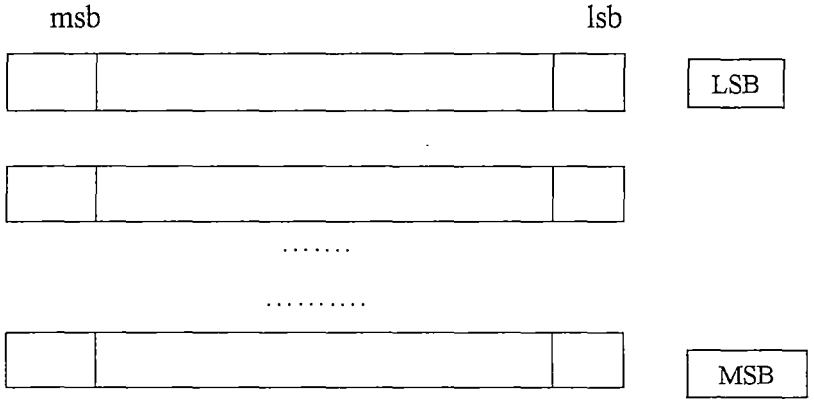

Fig 5.4 : n byte string stored in memory

Byte stored in the higher memory address has been considered as higher order byte in the string. So LSB is stored in lower memory address and MSB in higher memory address. The algorithm of 'lsbinv' is given in the following.

1) The BC pair, HL pair, DE pair is pushed in the stack.
2) The HL pair, used as memory pointer is pointed to f900h
3) The register C is loaded with the count value.
4) Register B is initialized with the masking byte (00000001b).
5) The memory content, pointed by HL, is tested for its lsb whether 0 or 1.
6) If it were zero, go to (7), for setting it 1.
   Else, reset it to 0 and go to (8).
7) The ls bit is set to 1
8) The memory pointer is incremented
9) The counter C is decremented.
10) Till the counter value is zero, go to (5).
11) The registers are popped back.
12) Return

By changing the counter value, the bit length of the string can be changed. Here, only the lsb has considered, to be inverted. However any bit in the bytes of the string can be inverted by changing the masking byte in register B.

5.6.2 Algorithm of 'rot'

This routine rotates the string anticlockwise by one bit, containing n bytes. Referring the fig 5.4, it is assumed that the string is stored from f900h onwards, LSB in f900h. The ls bit of the string stored in f900h is checked for 0 or 1 and set the carry accordingly. Then, the memory pointer is set to the last location, containing the MSB, and rotates the byte right by one bit through carry. The process is continued till the first location, containing the LSB, is reached.

1) Register C is initialized as counter
2) HL pair, used as memory pointer, is set to F900h
3) The memory content is moved to A.
4) 01h is ANDed with A.
5) If the zero flag is set, register B is loaded with 00h, otherwise with 01h.
6) The memory pointer is set to the last location.
7) The ls bit in B is shifted to carry bit.

8) The memory content is rotated through carry.

9) The pointer is decremented.

10) The byte counter, C is decremented.

11) Till the counter is exhausted, go to (8).

12) Returned.

By changing the count value in C, the bit length in String can be changed.

5.6.3 Algorithm 'store'

This routine is used for storing the string as well as the intermediate string generated from F900h onwards during encoding or decoding. Here the HL pair is used the pointer of the memory from where the bytes will stored. The initialization of the HL pair is made through the main program and will be used as parameter to the routine 'store'.

1) The BC and DE pair is saved in the stack.

2) The D is initialized with byte counter.

3) The BC pair is initialized with f900h.

4) The content of memory pointed by BC pair is moved to A.

5) The content of A is moved to the memory pointed by HL pair.

6) The HL and BC pair are incremented.

7) The D, byte counter is decremented.

8) Till it is zero, go to (4).

9) BC and DE pairs are incremented.

10) Returned.

By changing the counter value in D, the byte length can be changed..

5.6.4 Main Program

The main program is calling the routines developed in section 5.7.1 to 5.7.3 to encode the string using modified rotational encoding as given below.

1) The stack pointer is initialized at ffa0h.

2) The HL pair is initialized with fa00h.

3) The routine 'store' is called.

4) The routine 'lsbinv' is called.

5) The routine 'store' is called.

6) The counter C is loaded with the value, the number of rotations to be given during the encoding.

7) The routine 'rot' is called.

8) The counter is decremented.

9) Till it is zero, go to (7).

10) The routine 'lsbinv' is called.

11) The routine 'store' is called.

12) End.

5.7 Encoding

Here, the routine 'lsbinv' is called once and routine 'rot' is called j, the number of rotations to be given during the encoding, times. Where j is less than 8 times n, the number of bytes. The output generated at the memory where the string was supplied.

1) The stack pointer is initialized at ffa0h.

2) The routine 'lsbinv' is called.

3) The counter C is loaded with the value, the number of rotations to be given during the encoding.

4) The routine 'rot' is called.

5) The counter is decremented.

6) Till it is zero, go to (4).

7) End.

5.8 Decoding

Here, the routine 'rot' is called (8 n – j), the number of rotations to be given during the decoding, times and routine 'lsbinv' is called once.

1) The stack pointer is initialized at ffa0h.

2) The counter C is loaded with the value, the number of rotations to be given during the decoding

3) The routine 'rot' is called.

4) The counter is decremented.

5) Till it is zero, go to (3).

6) The routine 'lsbinv' is called.

7) End.

5.9 Result

The following data for different string length are given for presentation here. Table 5.4 shows the Read/Write memory required for testing a string of different bit length.

Table 5.4: Memory requirement to store the generated data

String length (bit)	String length (byte)	Number of rotation	Total number of bytes reqd. to store
16	2	16	32
32	4	32	128
64	8	64	512
128	16	128	2048
256	32	256	8192

For a system having 1k RAM available for storing intermediate data, the rotational encoding can be run on a string of 64bit length maximum. Because it takes 512 bytes for its storing the string and its intermediate strings. For 128 and 256bit string length, a special care, as in the following, has to be taken with same microprocessor based system, since it requires 2048 and 8192 bytes respectively.

i) The lsb inversion (routine 'lsbinv') will be called once on the string separately.

ii) Then the rotational encoding (routine 'rot') will be called 2 times with 64 rotations each for 128 bit (8 times with 32 rotations each for 256 bit) separately and the intermediate strings generated will be stored to different files and concatenated.

iii) The lsb inversion (routine 'lsbinv') will be called once again on the string generated last.

So the main routine will only be changed for calling the routines for 128/256 bit string. For a 16 bit string (50C8h), the least significant byte (C8h) is stored in lower memory and the most significant byte (50h) is stored in next location. After the inversion of two bits in two locations, the string is (51C9h). It is shown in table 5.5.

Table 5.5: lsb inversion in 16 bit string

String (16 bit) (hex)	String after lsb inversion (hex)	String recovered on lsb inversion again (hex)
50C8	51C9	50C8

The table 5.6 shows the Intermediate string generated for a 16 bit string under Rotational encoding.

Table 5.6: Intermediate string generated for a 16 bit string Under Rotational encoding

Rotation	String (hex)
After 0^{th} rotation	51C9
After 1^{st} rotation	A8E4
After 2^{nd} rotation	5472
After 3^{rd} rotation	2A39
After 4^{th} rotation	951C
After 5^{th} rotation	4A8E
After 6^{th} rotation	2547
After 7^{th} rotation	92A3
After 8^{th} rotation	C951
After 9^{th} rotation	E4A8
After 10^{th} rotation	7254
After 11^{th} rotation	392A
After 12^{th} rotation	1C95
After 13^{th} rotation	8E4A
After 14^{th} rotation	4725
After 15^{th} rotation	A392
After 16^{th} rotation (string recovered)	51C9

The lsb inversion is shown in table 5.7 for 32 bit string.

Table 5.7: lsb inversion for a 32 bit string

String (hex)	String after lsb inversion (hex)	String recovered on 2nd lsb inversion (hex)
61D950C8	60D851C9	61D950C8

The table 5.8 shows the intermediate strings generated for a 32 bit string.

Table 5.8: Intermediate string generated for a 32 bit string

Rotation	String (hex)	Rotation	String (hex)
After 0th rotation	60D851C9		
After 1st rotation	B06C28E4	After 17th rotation	28E4B06C
After 2nd rotation	58361472	After 18th rotation	1472 5836
After 3rd rotation	2C1B0A39	After 19th rotation	0A392C1B
After 4th rotation	960D851C	After 20th rotation	851C960D
After 5th rotation	4B06C28E	After 21st rotation	C28E4B06
After 6th rotation	25836147	After 22nd rotation	61472583
After 7th rotation	92C1B0A3	After 23rd rotation	B0A3 92C1
After 8th rotation	C960D851	After 24th rotation	D851 C960
After 9th rotation	E4B06C28	After 25th rotation	6C28 E4B0
After 10th rotation	72583614	After 26th rotation	36147258
After 11th rotation	392C1B0A	After 27th rotation	1B0A392C
After 12th rotation	1C960D85	After 28th rotation	0D851C96
After 13th rotation	8E4B06C2	After 29th rotation	06C28E4B
After 14th rotation	47258361	After 30th rotation	83614725
After 15th rotation	A392C1B0	After 31st rotation	C1B0A392
After 16th rotation	51C960D8	After 32nd rotation (string recovered)	60D851C8

For 64 bit string the data given in the table 4.5 are 8 byte data and are to be read LS byte first consisting of first two characters, the next two characters for next to LS

byte, and MS byte at last. For example, the 64 bit string (4BFB72EA 61D950C8h) is represented as (C850D961 EA72FB4B h).

The lsb inversion for 64 bit is shown in table 5.9. The table 5.10 shows the intermediate string generated for 64 bit string under rotational encoding.

Table 5.9: lsb inversion for 64 bit string

String (hex)	String after lsb inversion (hex)	String recovered on 2^{nd} lsb inversion (hex)
C850D961EA72FB4B	**C951D860EB73FA4A**	**C850D961EA72FB4B**

Table 5.10: Intermediate string generated for a 64 bit string Under Rotational encoding

Rotation	String (hex)	Rotation	String (hex)
0	**C951D860EB73FA4A**		
1	**E4286CB0F5397DA5**	33	**F5397DA5E4286CB0**
2	**721436D8FA9CBE52**	34	**FA9CBE52721436D8**
3	**390A1B6C7D4E5F29**	35	**7D4E5F29390A1B6C**
4	**1C850DB63EA7AF94**	36	**3EA7AF941C850DB6**
5	**8EC2065B9FD3574A**	37	**9FD3574A8EC2065B**
6	**476183ADCFE92B25**	38	**CFE92B25476183AD**
7	**A3B0C1D6E7F49592**	39	**E7F49592A3B0C1D6**
8	**51D860EB73FA4AC9**	40	**3FA4AC951D860EB3**
9	**286CB0F5397DA5E4**	41	**97DA5E4286CB0F59**
10	**1436D8FA9CBE5272**	42	**CBE52721436D8FA2**
11	**0A1B6C7D4E5F2939**	43	**4E5F29390A1B6C7D**
12	**850DB63EA7AF941C**	44	**A7AF941C850DB63E**
13	**C2065B9FD3574A8E**	45	**D3574A8EC2065B9F**
14	**6183ADCFE92B2547**	46	**E92B25476183ADCF**
15	**B0C1D6E7F49592A3**	47	**F49592A3B0C1D6E7**
16	**D860EB73FA4AC951**	48	**FA4AC951D860EB73**
17	**6CB0F5397DA5E428**	49	**7DA5E4286CB0F539**
18	**36D8FA9CBE527214**	50	**BE52721436D8FA9C**
19	**1B6C7D4E5F29390A**	51	**5F29390A1B6C7D4E**
20	**0DB63EA7AF941C85**	52	**AF941C850DB63EA7**
21	**065B9FD3574A8EC2**	53	**574A8EC2065B9FD3**
22	**83ADCFE92B254761**	54	**2B25476183ADCFE9**
23	**C1D6E7F49592A3B0**	55	**9592A3B0C1D6E7F4**
24	**60EB73FA4AC951D8**	56	**4AC951D860EB73FA**
25	**B0F5397DA5E4286C**	57	**A5E4286CB0F5397D**
26	**D8FA9CBE52721436**	58	**52721436D8FA9CBE**
27	**6C7D4E5F29390A1B**	59	**29390A1B6C7D4E5F**
28	**B63EA7AF941C850D**	60	**941C850DB63EA7AF**
29	**5B9FD3574A8EC206**	61	**4A8EC2065B9FD357**
30	**ADCFE92B25476183**	62	**25476183ADCFE92B**
31	**D6E7F49592A3B0C1**	63	**92A3B0C1D6E7F495**
32	**EB73FA4AC951D860**	64	**C951D860EB73FA4A**

For 128 bit string the data given in the table 5.12 are 16 byte data and are to be read ls byte first consisting of first two characters, the next two characters for next to LS byte, and ms byte at last. For example, the 128 bit string (4BFB72EA 61D950C8 4BFB72EA 61D950C8h) is represented as (C850D961EA72FB4B C850D961EA72FB4Bh)
The lower part of the string has to be concatenated to the right of the first to make the string, and then it is to be read as directed. The lsb inversion is shown in table 5.11.

Table 5.11: lsb inversion for 128bit string

String (hex)	String after lsb inversion (hex)	String recovered on 2nd lsb inversion (hex)
C850D961EA72FB4B C850D961EA72FB4B	C951D860EB73FA4A C951D860EB73FA4A	C850D961EA72FB4B C850D961EA72FB4B

Table 5.12: Intermediate string generated for a 128 bit string
Under Rotational encoding

Rotn	String	Rotn	String
0	0102030405060708090A0B0C0D0E0F10		
1	00810182028303840485058606870788	33	028303840485058606870788008101 82
2	80C0004181C1014282C2024383C30344	34	81C1014282C2024383C3034480C00041
3	406080A0C0E00021416181A1C1E10122	35	C0E00021416181A1C1E10122406080A0
4	203040506070809 0A0B0C0D0E0F00011	36	60708090A0B0C0D0E0F0001120304050
5	101820283038404850586068707880 08	37	303840485058606870788008101820 28
6	080C1014181C2024282C3034383C4004	38	181C2024282C3034383C4004080C1014
7	0406080A0C0E10121416181A1C1E2002	39	0C0E10121416181A1C1E20020406080A
8	0203040506070809 0A0B0C0D0E0F1001	40	060708090A0B0C0D0E0F100102030405
9	810182028303840485058606870788 00	41	830384048505860687078800810182 02
10	C0004181C1014282C2024383C3034480	42	C1014282C2024383C3034480C0004181
11	6080A0C0E00021416181A1C1E1012240	43	E00021416181A1C1E10122406080A0C0
12	30405060708090A0B0C0D0E0F0001120	44	708090A0B0C0D0E0F000112030405060
13	18202830384048505860687078800810	45	38404850586068707880081018202830
14	0C1014181C2024282C3034383C400408	46	1C2024282C3034383C4004080C101418
15	06080A0C0E10121416181A1C1E200204	47	0E10121416181A1C1E20020406080A0C
16	0304050607080 90A0B0C0D0E0F100102	48	0708090A0B0C0D0E0F10010203040506
17	018202830384048505860687078800 81	49	038404850586068707880081018202 83
18	004181C1014282C2024383C3034480C0	50	014282C2024383C3034480C0004181C1
19	80A0C0E00021416181A1C1E101224060	51	0021416181A1C1E10122406080A0C0E0
20	405060708090A0B0C0D0E0F000112030	52	8090A0B0C0D0E0F00011203040506070
21	202830384048505860687078800810 18	53	40485058606870788008101820283038
22	1014181C2024282C3034383C4004080C	54	2024282C3034383C4004080C1014181C
23	080A0C0E10121416181A1C1E20020406	55	10121416181A1C1E20020406080A0C0E
24	0405060708090A0B0C0D0E0F10010203	56	08090A0B0C0D0E0F1001020304050607
25	820283038404850586068707880081 01	57	840485058606870788008101820283 03
26	4181C1014282C2024383C3034480C000	58	4282C2024383C3034480C0004181C101
27	A0C0E00021416181A1C1E10122406080	59	21416181A1C1E10122406080A0C0E000
28	5060708090A0B0C0D0E0F00011203040	60	90A0B0C0D0E0F000112030405060708 0
29	283038404850586068707880081018 20	61	485058606870788008101820283038 40
30	14181C2024282C3034383C4004080C10	62	24282C3034383C4004080C1014181C20
31	0A0C0E10121416181A1C1E2002040608	63	121416181A1C1E20020406080A0C0E10
32	05060708090A0B0C0D0E0F1001020304	64	090A0B0C0D0E0F100102030405060708

Table 5.12: Continued

Rotn	String	Rotn	String
65	048505860687078800810182028303840	97	0687078800810182028303840485058
66	82C2024383C3034480C0004181C10142	98	83C3034480C0004181C1014282C20243
67	416181A1C1E10122406080A0C0E00021	99	C1E10122406080A0C0E00021416181A1
68	A0B0C0D0E0F000112030405060708090	100	E0F000112030405060708090A0B0C0D0
69	505860687078800810182028303840	101	70788008101820283038404850586068
70	282C3034383C4004080C1014181C2024	102	383C4004080C1014181C2024282C3034
71	1416181A1C1E20020406080A0C0E1012	103	1C1E20020406080A0C0E10121416181A
72	0A0B0C0D0E0F100102030405060708090	104	0E0F100102030405060708090A0B0C0D
73	85058606870788008101820283038404	105	87078800810182028303840485058606
74	C2024383C3034480C0004181C1014282	106	C3034480C0004181C1014282C2024383
75	6181A1C1E10122406080A0C0E0002141	107	E10122406080A0C0E00021416181A1C1
76	B0C0D0E0F000112030405060708090A0	108	F000112030405060708090A0B0C0D0E0
77	586068707880081018202830384048	109	78800810182028303840485058606870
78	2C3034383C4004080C1014181C202428	110	3C4004080C1014181C2024282C303438
79	16181A1C1E20020406080A0C0E101214	111	1E20020406080A0C0E10121416181A1C
80	0B0C0D0E0F100102030405060708090A	112	0F100102030405060708090A0B0C0D0E
81	05860687078800810182028303840485	113	07880081018202830384048505860687
82	024383C3034480C0004181C1014282C2	114	034480C0004181C1014282C2024383C3
83	81A1C1E10122406080A0C0E000214161	115	0122406080A0C0E00021416181A1C1E1
84	C0D0E0F000112030405060708090A0B0	116	00112030405060708090A0B0C0D0E0F0
85	60687078800810182028303840485058	117	80081018202830384048505860687078
86	3034383C4004080C1014181C2024282C	118	4004080C1014181C2024282C3034383C
87	181A1C1E20020406080A0C0E10121416	119	20020406080A0C0E10121416181A1C1E
88	0C0D0E0F100102030405060708090A0B	120	100102030405060708090A0B0C0D0E0F
89	860687078800810182028303840485	121	88008101820283038404850586068707
90	4383C3034480C0004181C1014282C202	122	4480C0004181C1014282C2024383C303
91	A1C1E10122406080A0C0E00021416181	123	22406080A0C0E00021416181A1C1E101
92	D0E0F000112030405060708090A0B0C0	124	1120304050607080 90A0B0C0D0E0F000
93	68707880081018202830384048505860	125	0810182028303840485058606870788 0
94	34383C4004080C1014181C2024282C30	126	04080C1014181C2024282C3034383C40
95	1A1C1E20020406080A0C0E1012141618	127	020406080A0C0E10121416181A1C1E20
96	0D0E0F100102030405060708090A0B0C	128	0102030405060708090A0B0C0D0E0F10

For **256 bit string, the Rotational Encoding** requires 256 operations to get back the string and 8k byte of memory to store the string and its intermediate strings. Obviously, the microprocessor based system with approximately1k available RAM for storing the result needs to operate 8 times, store the result in separate files and concatenate the files. For clarity, the result for 256 bit string is not presented here, but the strings are given in Appendix in Intel hex format. The table 5.13 shows a few intermediate strings after 64 operations and the table 5.14 shows the lsb inversion for 256 bit string. To read any string of multi-byte, the Least Significant byte is placed first and the most significant byte at last.

Table 5.13: 256 bit string and a few intermediate strings

Operation	256 bit string (hex)
After 0^{th} operation	0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
After 64 operation	090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F200102030405060708
After 128 operation	111213141516171819 1A1B1C1D1E1F2001020304050607080 90A0B0C0D0E0F10
After 192 operation	191A1B1C1D1E1F2001020304050607080 90A0B0C0D0E0F101112131415161718
After 256 operation	0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20

Table 5.14: lsb inversion for 256 bit string

String	0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
String after 1^{st} Lsb inversion	00030205040706090 80B0A0D0C0F0E11101312151417161918 1B1A1D1C1F1E21
String after 2^{nd} Lsb inversion	0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20

5.10 Memory Efficient Algorithm

The program, developed in assembly level up to 256bit length string, for rotational encoding, does not require any other memory as scratch pad during the time of execution. The generated string will be available from the memory area, where the string is supplied.

The program will successfully be run for higher bits, even the total message as a string, provided the memory permits to accommodate the message.

Hence it will be a **memory efficient program** for encoding.

5.11 Application of Modified Rotational Encoder

The Modified Rotational (MR) Encoder can be successfully utilized in encryption. The encoder is tested for 16, 32, 64, 128 and 256 bit length and it takes

16, 32, 64, 128 and 256 operations respectively. Any intermediate string can be used for encrypted string.

So a message can be considered as the binary strings, multiple of 8, concatenated altogether. The MR encoding can be applied on the concatenated strings, say x times which is less than the number of operations required to get back the string, at the time of encryption by a person interested to secure the message. The encrypted message will be sent to the recipient. The rest of the operations (n-x) is required be applied on the encrypted message, by the recipient to get back the message, where n is the bit length of the string.

The encoder has been tested and realized up to 256 bit block length. The brute force attack with this high block length will not be successful in decoding the encoded message under MR encoder.

5.12 Comparison of MR Encoder

A comparative study of MR Encoding with RSA Encoding has been made in order to judge the developed encoding. For this purpose two methods are adopted as in the following as in earlier encoders.

1. Frequency Distribution of Encoded file with RPP Encoding
2. Homogeneity Test (Chi-square) by a statistical method

The Frequency Distribution and Homogeneity Test presentations are given in section 5.10.1 and 5.10.2 respectively.

5.12.1 Frequency Distribution of Encoded file

To find out the frequency distribution of the technique, a text file (prt.txt, size: 11351 Bytes) is taken. The Modified Rotational Encoding technique has been applied and then the distribution of the characters of the text file and that of encoded file is plotted in figure 5.5. The distribution of the characters of the text file and that of encoded file with RSA is plotted in figure 5.6. The blue line in the bar graph shows the distribution for the text file, while the red line for encoded file for both figures 5.5 and 5.6. From the frequency distribution graph it is clear that frequencies are distributed through complete range of characters (0 to 255) in both proposed MRE and RSA techniques. Hence it may be inferred that the proposed technique may obtain a good security.

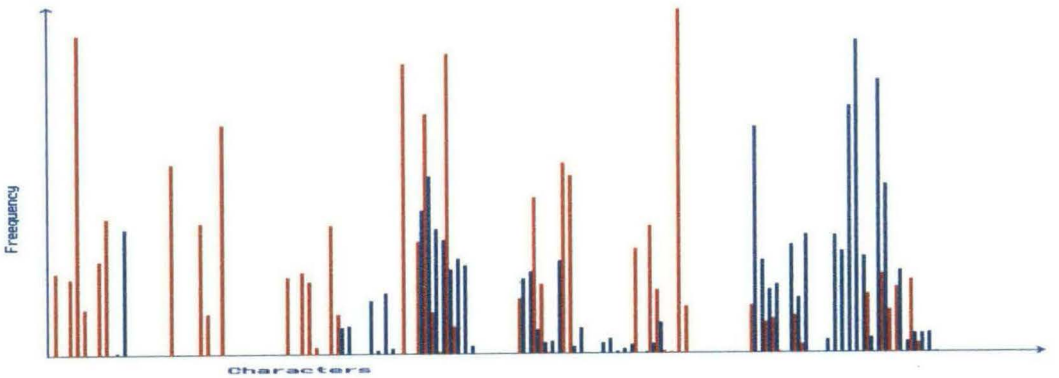

Figure 5.5: Frequency Distribution of characters in source message and encrypted message under Modified Rotational Encoding

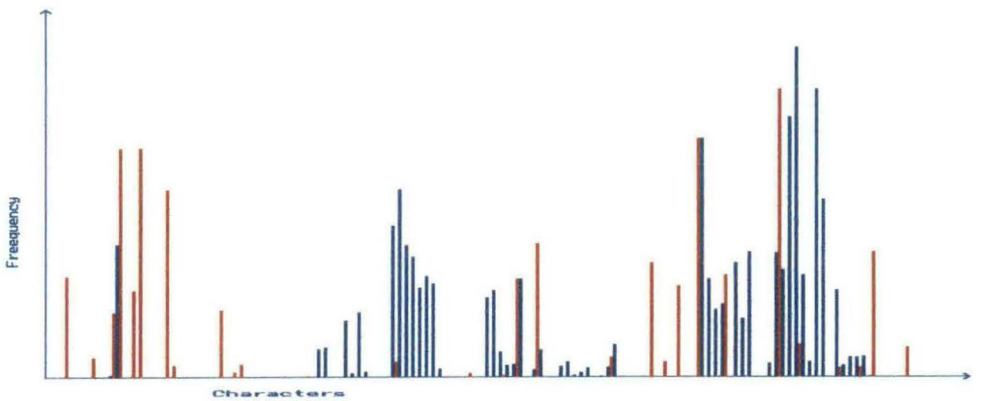

Figure 5.6: Frequency Distribution of characters in source message and encrypted message under RSA Encoding

5.12.2 Test of Homogeneity

For homogeneity test of the encrypted file generated through Rotational Encoder, the Chi-square method, a statistical procedure has been applied. A text file (g.abc) of size, 2157 byte is used for encryption. The corresponding encrypted files are generated for different block length. For each case the Chi-square value is computed. The maximum value for this text file is calculated as 3117.07 and the

average value of all the values is 2747.76. The table 5.15 shows the Chi-Square value and the figure 5.7 shows the diagram for different block lengths under Modified Rotational Encoder.

Table 5.15: Chi-square value for different Block Length of MR Encoder

Sl.No.	Block Length	Text File	Encrypted File	File Size	Chi-Square value
1	8	g.abc	Rt8.abc	2,157	2278.70
2	16	g.abc	Rt16.abc	2,157	3110.47
3	32	g.abc	Rt32.abc	2,157	2738.96
4	64	g.abc	Rt64.abc	2,157	3097.51
5	128	g.abc	Rt128.abc	2,157	2143.86
6	256	g.abc	Rt256.abc	2,157	3117.07
				Average Chi-Index	2747.76

The Ch-Square value computed on the same file under RSA is **2359.03**. The chi-square values for 16, 32, 64 and 256 bit block length under MR Encoder are higher than that of RSA Encoder. It is also shown in figure 5.7.

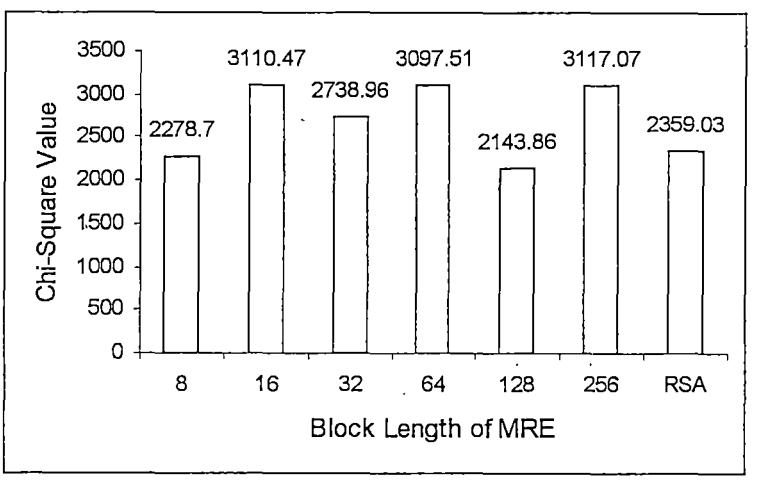

Figure 5.7: Chi-Square values for different Block length of Rotational Encoder

The chi-square value is also computed on ten files with different file size under MR Encoder as given in the table 5.16. Figure 5.8 gives the comparative study of the chi-square value for different block length under MR Encoder. Table 5.17 shows the average value of chi-square under RSA Encoder on the same files. From the average value computed, it is seen that for all the block length considered the

average chi-square values under MR Encoder exceed that of RSA and the over-all average value under MR Encoder is much higher than that of RSA.

Table 5.16: Chi-square value of MR Encoder for Different Block Length

Modified Rotational Encoder									
Sl no	Source File	Source File size	Chi-square For 8 bit length	Chi-square For 16 bit length	Chi-square For 32 bit length	Chi-square For 64 bit length	Chi-square For 128bit length	Chi-square For 256bit length	Ave Chi-Square
1	a.abc	904	1109.35	1316.06	1136.07	1284.01	1098.57	1086.26	
2	b.abc	1061	1471.86	1615.00	1441.31	1605.69	1435.07	1308.72	
3	c.abc	907	1258.83	1370.76	1237.29	1361.99	1248.86	1091.92	
4	d.abc	2841	3515.57	4142.43	3897.72	4147.48	3601.20	3407.21	
5	e.abc	1765	2294.55	2602.47	2455.67	2638.70	2317.90	2194.66	
6	f.abc	2227	2887.01	3339.16	3140.01	3376.64	3005.89	2769.38	
7	g.abc	2157	2760.56	3110.47	3000.47	3097.51	2842.84	2596.32	
8	h.abc	7121	8930.41	10587.39	9835.08	10544.42	9071.24	8496.01	
9	i.abc	8830	10944.10	13039.84	12041.59	13025.41	11049.97	10651.96	
10	j.abc	2182	2846.57	3288.61	3048.81	3316.18	2910.09	2670.99	
Average		2999.5	3801.88	4441.22	4123.40	4439.80	3858.16	3627.34	4048.63

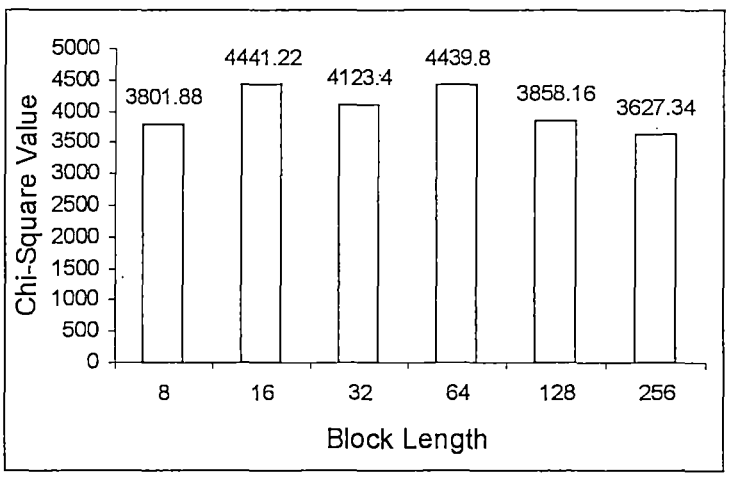

Figure 5.8: Chi-square value of RPPE for Different Block Length

Table 5.17: Chi-Square value under RSA Encoder

Sl no	Source File	Source File size	Chi-Square value under RSA Encoder
1	a.abc	904	953.21
2	b.abc	1061	1135.65
3	c.abc	907	997.82
4	d.abc	2841	2988.19
5	e.abc	1765	1892.01
6	f.abc	2227	2406.50
7	g.abc	2157	2359.03
8	h.abc	7121	7311.02

Sl no	Source File	Source File size	Chi-Square value under RSA Encoder
9	i.abc	8830	9085.07
10	j.abc	2182	2318.65
Average Chi-Square Value			**3144.715**

Table 5.18 shows chi-square values for different encoders. Figure 5.9 shows the pictorial diagram for the comparison of Chi-Square value of PP Encoder, Triangular Encoder (TE), RPP Encoder and MR Encoder with that of RSA Encoder. The Chi-square value against PPE is for overall average value of chi-square under Prime Position Encoder, that against PPE8 for average value of chi-square under Prime Position Encoder with 8 bit block length, that against TE for average value of Chi-square under Triangular Encoder, that against RPPE for average value of Chi-square under Recursive Pair Parity Encoder, that against MRE for average value of Chi-square under Modified Rotational Encoder and that against RSA for average value of RSA encoder.

Table 5.18: Comparison of Encoders on Chi-square value

Encoders	Chi-Square Value
PPE	1373.86
PPE8	3272.24
TE	3313.21
RPPE	3717.82
MRE	4048.63
RSA	3144.715

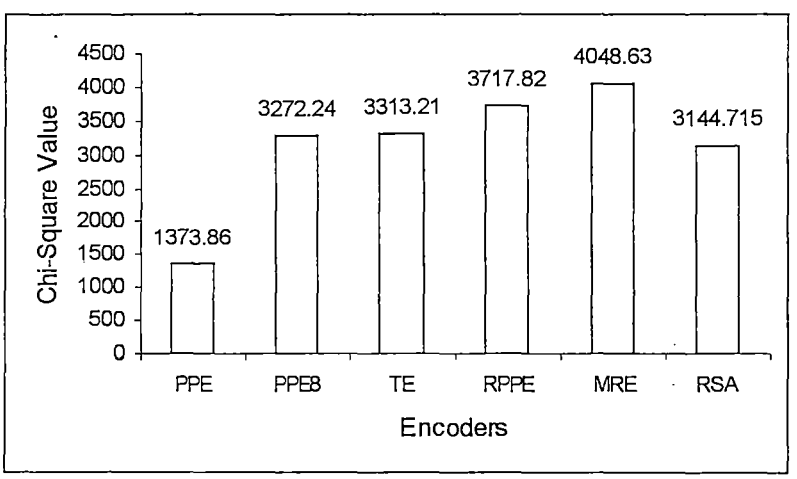

Figure 5.9: Comparison of Chi-Square value for different Encoders

The realized encoders are comparable with RSA with respect to Chi-square value. Higher chi-square value ensures the non-homogeneity of encrypted file with respect to source file. Hence the technique may lead to better security.

5.13 Conclusion

The Modified Rotational Encoder is a very simple in principle. It shows a very good frequency distribution of characters and the chi-square value in comparison with RSA encoder. The encryption time will be low making the operation of encoding and decoding faster.