Chapter - VI

Modified Johnson Encoder (MJE)

Chapter VI

Modified Johnson Encoding

Contents

6 Introduction

This is another encoding in tune with the Rotational Encoding. It is based on the principle of Johnson counter. The work is based on the block cipher technique. A block of binary data or string of a message is taken on which the transformation is applied. It generates the cipher text making the message in unintelligible form. This cipher text may be used as encrypted message for transmission being secure. Johnson encoder can generate 2n possible combinations for a string having n bits. Again n can be extended to a large value making the possible combinations large. This principle is utilized here for encoding.

6.1 Principle of Johnson Encoding

Considering an 8 bit string in binary as **(0 1 1 0 0 0 1 1).** According to Johnson Encoding the least significant bit (lsb) of the string is inverted and rotated right by one bit, transferring the lsb to msb position and all other bits shifted one bit to the right. The string generated will be **(0 0 1 1 0 0 0 1).** If this process applies successively on the same string, the original string will be generated after application of twice the number of bits on the string. The technique is explained for a block length of 2, 3 and 4 bits in section 6.1.1 to 6.1.3.

6.1.1 Johnson Encoder (JE) for 2 bit Block

A 2 bit concatenated string ($\mathbf{b_1}$ $\mathbf{b_0}$) is considered here, $\mathbf{b_1}$ being the msb and $\mathbf{b_0}$ the lsb. The underline of any bit indicates the inversion of the bit. After the first operation lsb ($\mathbf{b_0}$) is inverted and one rotation is applied on the string making it ($\underline{\mathbf{b_0}}$ $\mathbf{b_1}$). The same operation is applied on the string and the generated string is shown in table 6.1. It is seen that after 4 such operations the original string has been generated. So, for 2 bit case, the number of operations required to get back the string is 4.

Table 6.1: Johnson Encoding with 2 bit string

Concatenated string	No. of operation
$b_1\ b_0$	0
$\underline{b_0}\ b_1$	1
$\underline{b_1\ b_0}$	2
$b_0\ \underline{b_1}$	3
$b_1\ b_0$	4

6.1.2 JE for 3 bit Block

Here a 3 bit string has been considered. The same process is applied on the string. The intermediate string generated in the process of transformation is shown in table 6.2. It is seen that after 6 such operations the original string has been generated.

Table 6.2: Johnson Encoding with 3 bit string

Concatenated string	No. of operation
$b_2\ b_1\ b_0$	0
$\underline{b_0}\ b_2\ b_1$	1
$\underline{b_1\ b_0}\ b_2$	2
$\underline{b_2\ b_1\ b_0}$	3
$b_0\ \underline{b_2\ b_1}$	4
$b_1\ b_0\ \underline{b_2}$	5
$b_2\ b_1\ b_0$	6

So, for 3 bit case, the number of operations required to get back the string is 6.

6.1.3 JE for 4 bit Block

Here a 4 bit string has been considered. The same process is applied on the string. The intermediate string generated in the process of transformation is shown in table 6.3. It is seen that after 8 such operations the original string has been generated.

Table 6.3: Johnson Encoding with 4 bit string

Concatenated string	No. of operation
$b_3\ b_2\ b_1\ b_0$	0
$\underline{b_0}\ b_3\ b_2\ b_1$	1
$\underline{b_1}\ \underline{b_0}\ b_3\ b_2$	2
$\underline{b_2}\ \underline{b_1}\ \underline{b_0}\ b_3$	3
$\underline{b_3}\ \underline{b_2}\ \underline{b_1}\ \underline{b_0}$	4
$b_0\ \underline{b_3}\ \underline{b_2}\ \underline{b_1}$	5
$b_1\ b_0\ \underline{b_3}\ \underline{b_2}$	6
$b_2\ b_1\ b_0\ \underline{b_3}$	7
$b_3\ b_2\ b_1\ b_0$	8

So, for 4 bit case, the number of operations required to get back the string is 8.

In brief, the principle of Johnson encoding states that for a block of m bit string, the number of operation required is two times of m bit to get back the original string.

6.2 Modified Johnson Encoding

When a large number of bytes are taken into consideration in the string, the Johnson encoding will not be very effective. On 8^{th} rotation, the LS byte will go to the MS byte position with all its bits inverted and all other bytes will be shifted to the right. The characters in the string will appear again in the shifted condition except the MS byte. On 16^{th} rotation the same thing will happen with two MS bytes with all its bits inverted. 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 Johnson encoding.

This drawback must be eliminated otherwise it is not possible to use the technique in the encoding. To overcome the drawback of this encoder, the same precautionary measure as in rotational encoding has been applied. A particular bit of each byte in the string is inverted (here, it is lsb) first and the Johnson principle is applied. This is called Modified Johnson Encoding.

The Modified Johnson Encoding (MJE) can also be applicable to a string of any length. The encoding and decoding is discussed in section 6.2.1 and 6.2.2 respectively.

6.2.1 Modified Johnson Encoding

To encode a string, the following steps are to be followed.

i) Inversion of a bit (lsb) in each byte of the string

ii) Johnson Encoding on the string

To illustrate the encoding process an 8 bit binary string, **1011001** has been considered.

i) The String after Inversion of lsb is **10110010.**

ii) Johnson Encoding after k (here, it is 8) number of operations, where k is less than twice the number of bits is applied on the string as shown in table 6.4.

Table 6.4: 8 bit string after k operations under Johnson Encoding

No. of operation	Concatenated string
0	**1 0 1 1 0 0 1 0**
1	**1 1 0 1 1 0 0 1**
2	**0 1 1 0 1 1 0 0**
3	**1 0 1 1 0 1 1 0**
4	**1 1 0 1 1 0 1 1**
5	**0 1 1 0 1 1 0 1**
6	**0 0 1 1 0 1 1 0**
7	**1 0 0 1 1 0 1 1**
8	**0 1 0 0 1 1 0 1**

After 8 operations, the generated string **0 1 0 0 1 1 0 1** may be used as encoded string.

6.2.2 Modified Johnson Decoding

To decode the string the reverse steps are to be followed. First Johnson encoding for rest number of operations and then the lsb inversion are to be applied.

i) Johnson Decoding (same as Encoding) on the string for rest operations

ii) Inversion of a bit (lsb) in each byte of the string

To illustrate the decoding technique, the encoded 8 bit string **(0 1 0 0 1 1 0 1)** is considered for decoding. Since 8 operations are applied at the time of encoding, (2x8 – 8) i.e. 8 number of operations are required to be applied at the time of decoding. The eight operations are shown in table 6.5.

Table 6.5: Decoding of 8 bit string under Johnson Encoding

No. of operation	Concatenated string
0	**0 1 0 0 1 1 0 1**
1	**0 0 1 0 0 1 1 0**
2	**1 0 0 1 0 0 1 1**
3	**0 1 0 0 1 0 0 1**
4	**0 0 1 0 0 1 0 0**
5	**1 0 0 1 0 0 1 0**
6	**1 1 0 0 1 0 0 1**
7	**0 1 1 0 0 1 0 0**
8	**1 0 1 1 0 0 1 0**

Then the final string will be supplied for lsb inversion. It is shown in table 6.6.

Table 6.6: lsb inversion

String available from Johnson Decoder	String after lsb inversion
1 0 1 1 0 0 1 0	**1 0 1 1 0 0 1 1**

Hence, after the lsb inversion the original string is generated.

6.3 Modified Johnson Encoding for n byte string

The string consisting of n bytes is considered for Modified Johnson Encoding. The following steps are followed for the encoding.

1. Inversion of lsb of each byte of the string
2. Johnson Encoding on the string

Both these steps are followed during Encoding as well as Decoding. Step (1) is followed first, then step (2) at the time of Encoding, while step (2) is followed first, then step (1) at the time of Decoding.

The step (1) is a very simple technique, as discussed in section 6.2. Also it is known that the total number of Johnson Encoding operation for n byte string required to get back the original string is twice the number of bits in the string i.e. (2 x 8 x n). Any one of (2 x 8 x n – 1) possible number of string can be used as encoded string.

So at the time of Encoding, it is assumed that k number of Johnson encoding operations is applied, where k is less than (2 x 8 x n).

At the time of Decoding, (2 x 8 x n - k) number of operations will be applied.

6.4 Realization of Modified Johnson Encoder using Microprocessor Based System

To realize the Modified Johnson Encoder using Microprocessor Based System, three following routines are developed. These are

a) Routine 'lsbinv'

b) Routine 'rotj'

c) Routine 'store'

The routines 'lsbinv' and 'store' are the same as the routines used in Rotational Encoding in ChapterV. So the discussion of the algorithms of these routines is not given for clarity. The routine 'rotj' differs slightly from the routine 'rot' of chapter-V. In Rotational Encoding, the lsb takes the position of msb and other bits in the string moves one bit to the right, while in Johnson Encoding, the lsb is inverted first and takes the position of msb and other bits in the string moves one bit to the right. For example, a 4 byte string is shown in fig 6.1. The lsb will be inverted and put to the msb position, all other bits in the string will be shifted to the right by one bit.

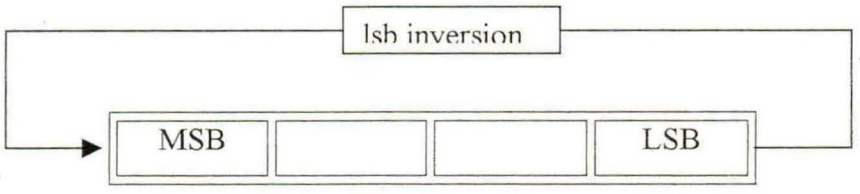

Fig 6.1: Rotational operation under Johnson Encoding

The routines to realize the MJE are given in section 6.4.1 to 6.4.2. Some routines are already discussed in section 5.6.1 and 5.6.3 in chapter V.

6.4.1 Algorithm of 'rotj'

This routine rotates the string anticlockwise by one bit, containing n bytes. 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 to its inverted bit accordingly. Then, the memory pointer is set to the last location, containing the MSB, and rotated 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 01h, otherwise with 00h.
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.

6.4.2 Main Program

The main program is calling the routines 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 'rotj' 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.

Up to 64 bit string, the main algorithm may be applicable for microprocessor based systems with 1k RAM in order to store the string and the intermediately generated strings. For 128/ 256 bit string, a special attention is required to store the strings generated with same system. It is discussed in the section 6.5.

6.5 Results

The table 6.7 shows the Read/Write memory required for testing a string of different bit length. For a system having 1k RAM available for storing intermediate data, the Johnson encoding can be run on a string of 64 bit length maximum effectively. Because it takes 1024 byte for storing the string and its intermediate strings. A special care, as in the following, has to be taken with same microprocessor based system for 128 and 256 bit string length, since it requires 4096 and 16394 bytes respectively.

Table 6.7: Memory required for the string of different bit length

String length (bit)	String length (byte)	Number of rotation	Total number of bytes reqd. to store
16	2	32	64
32	4	64	256
64	8	128	1024
128	16	256	4096
256	32	512	16394

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

ii) Then the rotational encoding (routine 'rot') will be called 4 **times with 64 rotations each for 128** bit (**16 times with 32 rotations each for 256** bit) separately and the intermediate strings generated will be stored to different files and concatenated.

Instead of storing all the strings for 256 bit case, the 'rotj' is called 4 times with 128 rotations, the sample strings collected and checked the final string as shown for 256 bit string.

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** (0102h), the least significant byte (01h) is stored in lower memory and the most significant byte (02h) is stored in next location. After the inversion of lsb of each byte, the string is (0003h). It is shown in table 6.8. The string 0201h is represented here as 0102h.

Table 6.8: lsb inversion of a 16 bit string under Johnson Encoding

String (16 bit) (hex)	String after lsb inversion (hex)	String recovered on lsb inversion again (hex)
0102h	0003h	0102h

The table 6.9 shows the Intermediate string generated for a 16 bit string under Johnson encoding.

Table 6.9: Intermediate string of 16 bit string under Johnson encoding

Rotation	String (hex)	Rotation	String (hex)
After 0^{th} rotation	0003		
After 1^{st} rotation	8081	After 17^{th} rotation	7F7E
After 2^{nd} rotation	C0C0	After 18^{th} rotation	3F3F
After 3^{rd} rotation	60E0	After 19^{th} rotation	9F1F
After 4^{th} rotation	30F0	After 20^{th} rotation	CF0F
After 5^{th} rotation	18F8	After 21^{st} rotation	E707
After 6^{th} rotation	0CFC	After 22^{nd} rotation	F303
After 7^{th} rotation	06FE	After 23^{rd} rotation	F901
After 8^{th} rotation	03FF	After 24^{th} rotation	FC00
After 9^{th} rotation	817F	After 25^{th} rotation	7E80
After 10^{th} rotation	C03F	After 26^{th} rotation	3FC0
After 11^{th} rotation	E09F	After 27^{th} rotation	1F60
After 12^{th} rotation	F0CF	After 28^{th} rotation	0F30
After 13^{th} rotation	F8E7	After 29^{th} rotation	0718
After 14^{th} rotation	FCF3	After 30^{th} rotation	030C
After 15^{th} rotation	FEF9	After 31^{st} rotation	0106
After 16^{th} rotation (string recovered)	FFFC	After 32^{nd} rotation (string recovered)	0003

The lsb inversion of each byte for 32 bit string is shown in table 6.10.

Table 6.10: lsb inversion of each byte for 32 bit string

String (hex)	String after lsb inversion (hex)	String recovered on 2^{nd} lsb inversion (hex)
01020304	00030205	01020304

The table 6.11 shows the intermediate strings generated for a 32 bit string under Johnson encoding.

Table 6.11: Intermediate string generated for a 32 bit string

Under Johnson encoding

Rotation	String (hex)	Rotation	String (hex)
0	00030205		
1	80018182	33	7FFE7E7D
2	C08040C1	34	3F7FBF3E
3	6040A0E0	35	9FBF5F1F
4	302050F0	36	CFDFAF0F
5	181028F8	37	E7EFD707
6	0C0814FC	38	F3F7EB03
7	06040AFE	39	F9FBF501
8	030205FF	40	FCFDFA00
9	0181827F	41	FE7E7D80
10	8040C13F	42	7FBF3EC0
11	40A0E09F	43	BF5F1F60
12	2050F0CF	44	DFAF0F30
13	1028F8E7	45	EFD70718
14	0814FCF3	46	F7EB030C
15	040AFEF9	47	FBF50106
16	0205FFFC	48	FDFA0003
17	81827FFE	49	7E7D8001
18	40C13F7F	50	BF3EC080
19	A0E09FBF	51	5F1F6040
20	50F0CFDF	52	AF0F3020
21	28F8E7EF	53	D7071810
22	14FCF3F7	54	EB030C08
23	0AFEF9FB	55	F5010604
24	05FFFCFD	56	FA000302
25	827FFE7E	57	7D800181
26	C13F7FBF	58	3EC08040
27	E09FBF5F	59	1F6040A0
28	F0CFDFAF	60	0F302050
29	F8E7EFD7	61	07181028
30	FCF3F7EB	62	030C0814
31	FEF9FBF5	63	0106040A
32	FFFCFDFA	64	00030205

For 64 bit string the data given in the table 6.13 are 8 byte data and are to be read ls byte first, and 8th byte as ms byte. For example, the 64 bit string (**0807060504030201**h) is represented as (**0102030405060708**h).

The lsb inversion for **64 bit** is shown in table 6.12.

Table 6.12: lsb inversion for 64 bit under Johnson Encoding

String (hex)	String after lsb inversion (hex)	String recovered on 2nd lsb inversion (hex)
0102030405060708	0003020504070609	0102030405060708

Table 6.13: Intermediate string generated for 64 bit string under Johnson encoding

Rotation	String (hex)	Rotation	String (hex)
0	0003020504070609		
1	8001810282038384	33	820383847FFE7EFD
2	C0804001C18141C2	34	C18141C23F7FBFFE
3	6040A080E0C020E1	35	E0C020E19FBF5F7F
4	30205040706090F0	36	706090F0CFDFAFBF
5	18102820383048F8	37	383048F8E7EFD7DF
6	0C0814101C1824FC	38	1C1824FCF3F7EBEF
7	06040A080E0C12FE	39	0E0C12FEF9FBF5F7
8	03020504070609FF	40	070609FFFCFDFAFB
9	018102820383847F	41	0383847FFE7EFD7D
10	804001C18141C23F	42	8141C23F7FBFFE3E
11	40A080E0C020E19F	43	C020E19FBF5F7F1F
12	205040706090F0CF	44	6090F0CFDFAFBF8F
13	102820383048F8E7	45	3048F8E7EFD7DFC7
14	0814101C1824FCF3	46	1824FCF3F7EBEFE3
15	040A080E0C12FEF9	47	0C12FEF9FBF5F7F1
16	020504070609FFFC	48	0609FFFCFDFAFBF8
17	8102820383847FFE	49	83847FFE7EFD7DFC
18	4001C18141C23F7F	50	41C23F7FBFFE3E7E
19	A080E0C020E19FBF	51	20E19FBF5F7F1F3F
20	5040706090F0CFDF	52	90F0CFDFAFBF8F9F
21	2820383048F8E7EF	53	48F8E7EFD7DFC7CF
22	14101C1824FCF3F7	54	24FCF3F7EBEFE3E7
23	0A080E0C12FEF9FB	55	12FEF9FBF5F7F1F3
24	0504070609FFFCFD	56	09FFFCFDFAFBF8F9
25	02820383847FFE7E	57	847FFE7EFD7DFC7C
26	01C18141C23F7FBF	58	C23F7FBFFE3E7EBE
27	80E0C020E19FBF5F	59	E19FBF5F7F1F3FDF
28	40706090F0CFDFAF	60	F0CFDFAFBF8F9F6F
29	20383048F8E7EFD7	61	F8E7EFD7DFC7CFB7
30	101C1824FCF3F7EB	62	FCF3F7EBEFE3E7DB
31	080E0C12FEF9FBF5	63	FEF9FBF5F7F1F3ED
32	04070609FFFCFDFA	64	FFFCFDFAFBF8F9F6

Table 6.13: Continued

Rotation	String (hex)	Rotation	String (hex)
65	7FFE7EFD7DFC7C7B	97	7F1F3FDF1E6040A0
66	3F7FBFFE3E7EBE3D	98	3E7EBE3DC0804001
67	9FBF5F7F1F3FDF1E	99	1F3FDF1E6040A080
68	CFDFAFBF8F9F6F0F	100	8F9F6F0F30205040
69	E7EFD7DFC7CFB707	101	C7CFB70718102820
70	F3F7EBEFE3E7DB03	102	E3E7DB030C081410
71	F9FBF5F7F1F3ED01	103	F1F3ED0106040A08
72	FCFDFAFBF8F9F600	104	F8F9F60003020504
73	FE7EFD7DFC7C7B80	105	FC7C7B8001810282
74	7FBFFE3E7EBE3DC0	106	7EBE3DC0804001C1
75	BF5F7F1F3FDF1E60	107	3FDF1E6040A080E0

Rota-tion	String (hex)	Rota-tion	String (hex)
76	DFAFBF8F9F6F0F30	108	9F6F0F3020504070
77	EFD7DFC7CFB70718	109	CFB7071810282038
78	F7EBEFE3E7DB030C	110	E7DB030C0814101C
79	FBF5F7F1F3ED0106	111	F3ED0106040A080E
80	FDFAFBF8F9F60003	112	F9F6000302050407
81	7EFD7DFC7C7B8001	113	7C7B800181028203
82	BFFE3E7EBE3DC080	114	BE3DC0804001C181
83	5F7F1F3FDF1E6040	115	DF1E6040A080E0C0
84	AFBF8F9F6F0F3020	116	6F0F302050407060
85	D7DFC7CFB7071810	117	B707181028203830
86	EBEFE3E7DB030C08	118	DB030C0814101C18
87	F5F7F1F3ED010604	119	ED0106040A080E0C
88	FAFBF8F9F6000302	120	F600030205040706
89	FD7DFC7C7B800181	121	7B80018102820383
90	FE3E7EBE3DC08040	122	3DC0804001C18141
91	BF8F9F6F0F302050	123	1E6040A080E0C020
92	DFC7CFB707181028	124	0F30205040706090
93	EFE3E7DB030C0814	125	0718102820383048
94	F7F1F3ED0106040A	126	030C0814101C1824
95	FBF8F9F600030205	127	0106040A080E0C12
96	7DFC7C7B80018102	128	**0003020504070609**

For **128bit string, the Johnson Encoding** requires 256 operations to get back the string and 4k 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 4 times, store the result in separate files and concatenate the files. For clarity, the result for 128 bit string is not presented here, and is given in Appendix in Intel Hex Format. The table 6.14 shows a few intermediate strings and the table 6.15 shows the lsb inversion for 128 bit string. To read any string of multi-byte, the Least Significant byte is placed first and the most significant byte at last.

Table 6.14: 128 bit string and a few intermediate string

Operation	**128 bit string (hex)**
After 0 operation	0102030405060708090A0B0C0D0E0F10
After 64 operation	090A0B0C0D0E0F10FEFDFCFBFAF9F8F7
After 128 operation	FEFDFCFBFAF9F8F7F6F5F4F3F2F1F0EF
After 192 operation	F6F5F4F3F2F1F0EF0102030405060708
After 256 operation	0102030405060708090A0B0C0D0E0F10

Table 6.15: lsb inversion for 128 bit string

String	0102030405060708090A0B0C0D0E0F10
String after 1st lsb inversion	00030205040706090B0A0D0C0F0E11
String after 2nd lsb inversion	0102030405060708090A0B0C0D0E0F10

For **256 bit string, the Johnson Encoding** requires 512 operations to get back the string and 16 k 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 16 times, store the result in separate files and concatenate the files. For clarity, the result for 256 bit string is not presented here and is given in Appendix. The table 6.16 shows a few intermediate strings after 128 operations and the table 6.17 shows the lsb inversion for 256 bit string. The results are also available in Appendix-V in Intel Hex Format. To read any string of multi-byte, the Least Significant byte is placed first and the most significant byte at last.

Table 6.16: 256 bit string and a few intermediate strings

Operation	256 bit string (hex)
After 0th operation	0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
After 128 operation	1112131415161718191A1B1C1D1E1F20FEFDFCFBFAF9F8F7F6F5F4F3F2F1F0EF
After 256 operation	FEFDFCFBFAF9F8F7F6F5F4F3F2F1F0EFEEEDECEBEAE9E8E7E6E5E4E3E2E1E0DF
After 384 operation	EEEDECEBEAE9E8E7E6E5E4E3E2E1E0DF0102030405060708090A0B0C0D0E0F10
After 512 operation	0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20

Table 6.17: lsb inversion for 256 bit string

String	0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
String after 1st Lsb inversion	00030205040706090B0A0D0C0F0E11101312151417161918 1B1A1D1C1F1E21
String after 2nd Lsb inversion	0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20

The results up to 256 bit string are presented after running the corresponding programs given in Appendix.

6.6 Memory Efficient Program:

The program, developed in assembly level up to 256bit length string, for Johnson does not require any other memory as scratch pad during the time of execution. The generated string will be available for 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 is also a **memory efficient program** for encoding.

6.7 Application of Modified Johnson Encoder

The Modified Johnson (MJ) 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 during iteration of a particular block length can be used for encrypted string.

A message can be considered as the binary strings, multiple of 8, concatenated altogether. The MJ 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 (2n-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 MJ encoder.

6.8 Comparison of Modified Johnson Encoder

A comparative study of MJ Encoding with RSA Encoding with RSA technique as well as with other devised encoders 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 MJ Encoding
2. Homogeneity Test (Chi-square) by a statistical method

The Frequency Distribution and Homogeneity Test presentations are given in section 6.8.1 and 6.8.2 respectively.

6.8.1 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 Johnson 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 6.2. The distribution of the characters of the text file and that of encoded file with RSA is plotted in figure 6.3. The blue line in the bar graph shows the distribution for the text file, while the red line for encoded file for both figures 6.2 and 6.3.

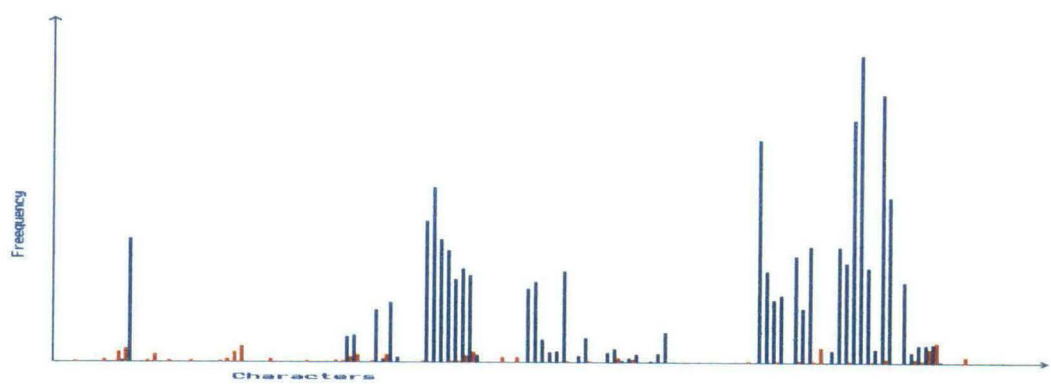

Figure 6.2: Frequency Distribution of characters in source message and encrypted message under Modified Johnson Encoding

From the frequency distribution graphs it is clear that the frequencies are distributed through complete range of characters (0 to 255) in both proposed MJE and

techniques. Hence it may be concluded that the proposed technique may provide a better security.

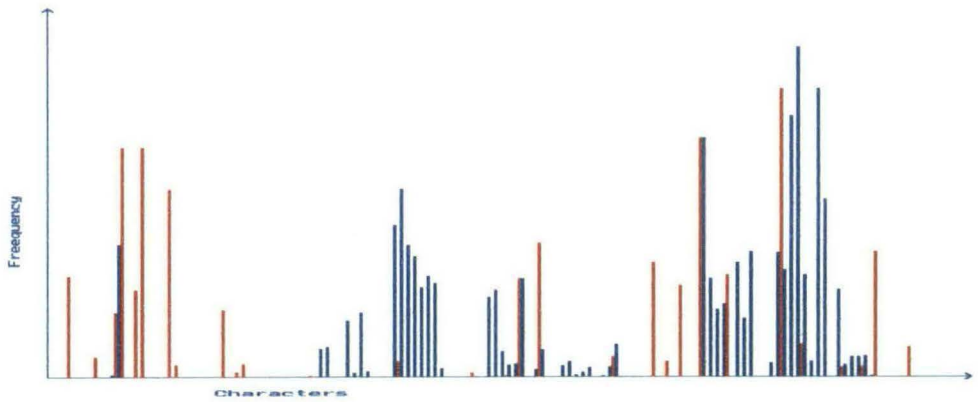

Figure 6.3: Frequency Distribution of characters in source message and encrypted message under RSA Encoding

6.8.2 Test of Homogeneity

For homogeneity test of the encrypted file generated through Johnson Encoder, 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 2313.47 and the average value of all the values is 1793.69. The table 6.18 shows the Chi-square value and the figure 6.4 shows the diagram for different block lengths.

Table 6.18: Chi-Indices for different Block Length of Johnson Encoder

Sl.No.	Block Length	Text File	Encrypted File	File Size	Chi-Index
1	8	g.abc	Jn8.abc	2,157	1897.42
2	16	g.abc	Jn16.abc	2,157	1955.75
3	32	g.abc	Jn32.abc	2,157	2313.47
4	64	g.abc	Jn64.abc	2,157	1860.15
5	128	g.abc	Jn128.abc	2,157	1021.25
6	256	g.abc	Jn256.abc	2,157	1714.09
				Average Chi-Index	1793.69

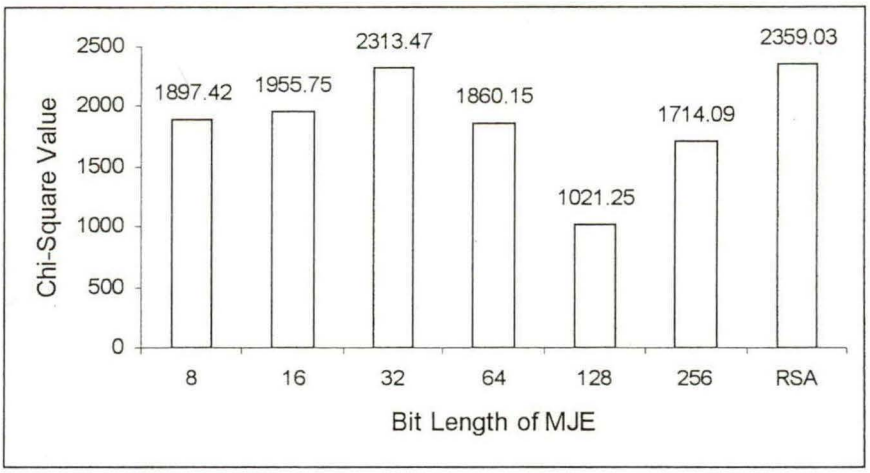

Figure 6.4: Chi-Square value for different Block length of Johnson Encoder

The chi-square value is also computed on ten files with different file size under MJ Encoder as given in the table 6.19. Figure 6.5 gives the comparative study of the chi-square value for different block length under MR Encoder. Table 6.20 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 MJ Encoder exceed that of RSA except the average value for 16 bit block length and the over-all average value under MJ Encoder is much higher than that of RSA.

Table 6.19: Chi-square value of MJ Encoder for Different Block Length

Johnson 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	Av Chi-Square (overall)
1	a.abc	904	1120.88	1182.92	1214.89	1242.94	1268.80	1273.19	
2	b.abc	1061	1428.69	1535.73	1583.92	1590.85	1597.17	1607.06	
3	c.abc	907	1226.82	1312.53	1347.78	1355.73	1369.26	1367.45	
4	d.abc	2841	3748.21	3860.95	4011.11	4096.90	4109.80	4123.25	
5	e.abc	1765	2324.76	2458.00	2527.23	2588.38	2600.17	2617.36	
6	f.abc	2227	2876.52	3095.78	3228.18	3310.49	3349.98	3354.83	
7	g.abc	2157	2816.41	2920.78	2998.06	3068.83	3076.86	3113.73	
8	h.abc	7121	9307.56	9868.90	10170.16	10365.23	10478.05	10531.60	
9	i.abc	8830	11687.59	1143.22	12600.84	12894.00	12986.56	12993.66	
10	j.abc	2182	2850.30	3051.06	3176.02	3247.08	3300.38	3309.84	
Average		2999.5	3938.77	3042.99	4285.82	4376.04	4413.70	4429.20	4081.09

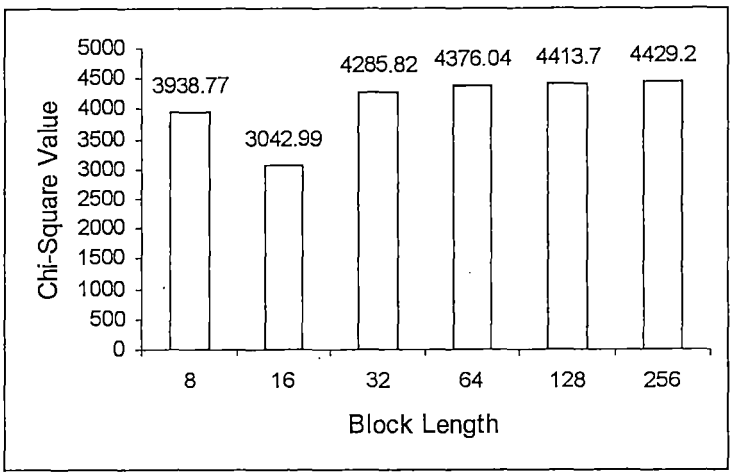

Figure 6.5: Chi-square value of MJ for Different Block Length

Table 6.20: 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
9	i.abc	8830	9085.07
10	j.abc	2182	2318.65
Average Chi-Square Value			**3144.715**

Table 6.21: 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
MJE	4081.09
RSA	3144.715

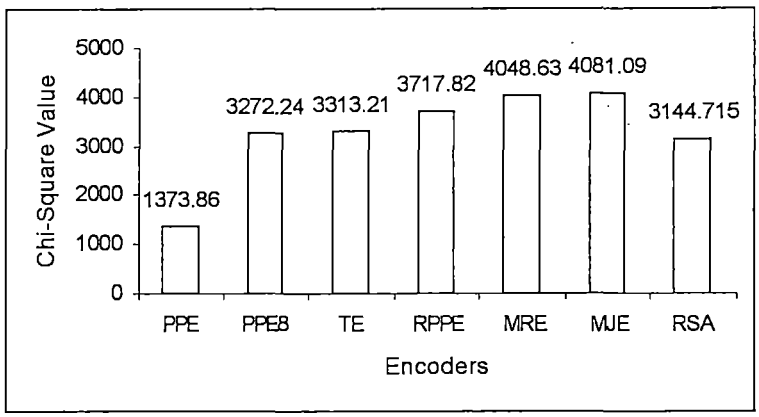

Figure 6.6: Comparison of Chi-Square value for different Encoders

Table 6.21 shows chi-square values for different encoders proposed along with RSA technique. Figure 6.6 shows the pictorial diagram for the comparison of Chi-Square value of PP Encoder, Triangular Encoder (TE), RPP Encoder, MR Encoder and MJ 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, that against MJE for average value of Chi-square under Modified Johnson Encoder and that against RSA for average value of RSA encoder.

The realized encoders are comparable with RSA with respect to Chi-square value.

6.9 Conclusion

The Modified Johnson Encoder is a very simple in principle. Higher chi-square values ensure the non-homogeneity of the encrypted file with respect to the source file. Hence it may be concluded that the proposed technique may offer a good security in encryption. 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.