Chapter 6

Text Encryption Building Blocks using Operators and Position of Bits in Plain Text

6.1. Overview

Sharing of the private key between sender and receiver without interpretation through open public communication channels is very hard to achieve. So the focus has been imposed to design a secret procedure which retrieves a secret value from the private key and applying that value for encryption rather than using the direct private key value. On that view, some schemes are newly implemented and have been discussed in chapter 5 where a single private key is applied for encrypting all the characters of a plain text file. So in this current chapter, some text encryption schemes based on operators, even odd terms and numbers of ‘0’,’1’ present in binary representation of plain text’s character are implemented where a separate private key is applied for encrypting each character of the plain text file.

Three text based encryption schemes are discussed in this chapter. They are Multiple Operator and Even Odd position based text encryption (MOEO)[1], Multiple Operator and ASCII Value based text encryption (MOAV)[2] and Multiple Operator and number of Zeros and Ones based text encryption (MOZO)[3]. Appreciable performances are measured in respect of encryption time, chi-square value and degree of freedom value.

In this chapter, Multiple Operator and Even Odd position based text encryption (MOEO) in section 6.2, Multiple Operator and ASCII Value based text encryption (MOAV) in section 6.3, Multiple Operator and number of Zeros and Ones based text encryption

(MOZO) in section 6.4 have been discussed. Conclusion regarding all the implemented schemes is attached in section 6.5.

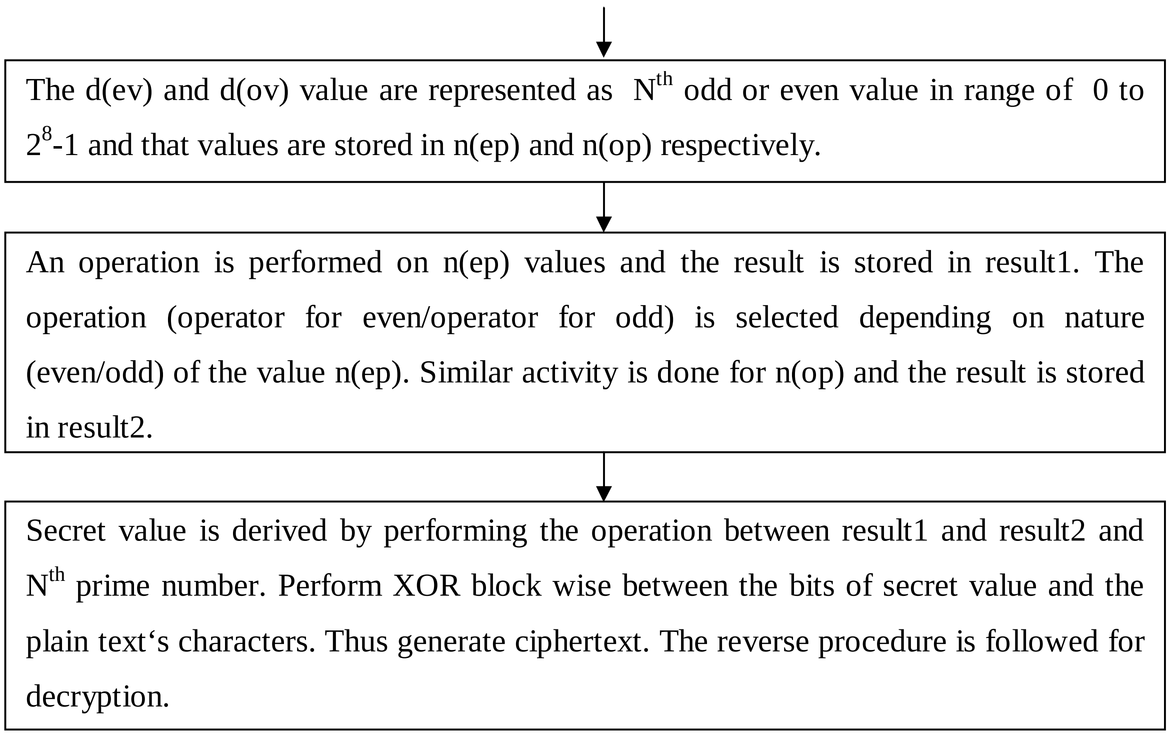
6.2. Multiple Operator and Even Odd position based text encryption scheme (MOEO)

In MOEO[1] scheme, both encryption and decryption are carried out by using the derived secret value from primary key rather than using the private key directly. Generation of secret value is done by performing user defined arithmetic operation between three decimal values where those decimal values are generated from the binary values stored in even and the odd bit position of a plain text's character respectively and Nth prime value (Where N is a positive integer with a range of 0 to 2^{62}-1). The operators are fetched from the corresponding blocks of private key and operators are inputted by the user. Different secret value is generated for encryption of each of the plain text's character as the formations of '0' and '1' are different for each distinct character in the plain text file. Thus provide great security .Figure 6.1 represents the overall procedure of Multiple Operator and Even Odd position based text encryption scheme(MOEO).

Formation of plain text is done by converting each character into an 8-bit binary representation. Generate private key by taking inputs from the user.

Two Decimal values, decimal value for even bit position (d (ev)) and decimal value for odd bit position (d (ov)) are generated by taking binary values of the even bit position and odd bit position of plain text's character respectively and N^{th} prime value is generated with a forward movement from user inputted base value (where N is a positive integer with a range of 0 to 2^{62}-1)

[1] Published in **International Journal of Engineering and Advanced Technology (IJEAT),** Volume 2, Issue 4, pp. 520-524,with title An approach of Private-key Encryption Technique based on Multiple Operators and Nth Even or Odd term for Even or Odd bit position's value of a Plain Text's character



The d(ev) and d(ov) value are represented as N th odd or even value in range of 0 to 2^8-1 and that values are stored in n(ep) and n(op) respectively.

An operation is performed on n(ep) values and the result is stored in result1. The operation (operator for even/operator for odd) is selected depending on nature (even/odd) of the value n(ep). Similar activity is done for n(op) and the result is stored in result2.

Secret value is derived by performing the operation between result1 and result2 and N th prime number. Perform XOR block wise between the bits of secret value and the plain text‘s characters. Thus generate ciphertext. The reverse procedure is followed for decryption.

Figure 6.1: Overall Procedure of Multiple Operator and Even Odd position based text encryption scheme(MOEO)

Section 6.2.1 and section 6.2.2 represents the encryption and decryption process respectively. Execution results are shown in section 6.2.3 and security analysis of the scheme are discussed in section 6.2.4.

6.2.1. Encryption Process

A. Formation of Plain Text

Step 1: Convert each character into 8-bit binary representation from plain text file till all characters are visited. Store the characters into an array called PT[].

B. Generation of Private Key

Step 1: Read user inputs for first, second, seventh, eight and ninth block of the primary key whereas values for third, fourth, fifth and six blocks of the key are computed from each character's value of the plain text.

Step 2: Convert the inputs into bits and computed values corresponding to their respective blocks of the primary key and store the values into an array called KEY[].

There are nine blocks in the private key where its size is 256 bits. First and second blocks hold the operators for even and odd values respectively where the value is generated from the even or odd bit position's binary value of a plain text's character. The third block holds the selection code which defines that the fourth block's content is even or odd. The fourth block represents a value in respect of N^{th} (N is a positive integer with a range of 0 to 2^8-1) even or odd term where the value is generated from the even position's bit value of plain text's character. The fifth block holds the selection code which defines that the sixth block's content is even or odd. The sixth block represents a value which is generated from the odd position's bit value of plain text 's character in respect of N^{th} (N is a positive integer with a range of 0 to 2^8-1) even or odd term. The base operator is stored in the seventh block. Eighth block stores the N^{th} (N is a positive integer with a range of 0 to 2^{62}-1) term for a prime number where base value is stored in the ninth block. Figure 6.2 represents the structure of the key.

1st block	2nd block	3rd block	4th block
Operator applied for even values o(ep)	Operator applied for odd values o(op)	Selection code for even or odd	Nth odd/even term for even position's bit value of plain text's single character. n(ep)
8 bits	8 bits	2 bits	8 bits

5th block	6th block	7th block	8th block	9th block
Selection code for even or odd	Nth odd/even term for odd position's bit value of plain text's single character. n(op)	Operator applied for base value O(b)	Nth term for prime number	Base value
2 bits	8 bits	8 bits	62 bits	150 bits

Figure 6.2: Structure of 256 bits Private Key

C. Generation of Secret Value for Encryption

Step 1: Decimal value from even bit position d(ev) and decimal value from odd bit position d(ov) are generated from the bit values present in even and the odd bit position of a plain text character respectively. Represent d(ev) and d(ov) in respect of N^{th} (N is a positive integer with a range of 0 to 2^8-1) even or odd term and store them in n(ep) and n(op) respectively.

.

Step 2: An operation specified by the operator (operator for even/operator for odd) is executed on n(ep) value depending upon nature (even/odd) of the value and the result is stored in result1. Similar activity is done for n(op) and the result is stored in result2. The N^{th} prime number is generated by making a forward movement from user defined base value and stored in result3.

Step 3: Base operator is executed between result1 and result2 and intermediate value is generated. Again the base operator is performed between intermediate value and result3 and secret value is generated.

Step 4: Secret value is converted into binary representation and stored into array DV[].

D. Generation of Ciphertext using XOR Operation

Step 1: Cumulative XOR operation is carried out between the array PT[] and DV[] and the resultant bits are stored in array EN[].

Step 2: ASCII value is generated from array EN[] and the corresponding encrypted character is generated from that ASCII value. Thus generates ciphertext file.

6.2.2. Decryption Process

A. Generation of Binary Representation of Ciphertext

Convert each character from the ciphertext file into 8-bit binary representation and store it into an array called CT[].

B. Generation of Secret Value for Decryption

Derive secret value for decryption by using section 6.2.1.C and store the bit values in an array called DV[].

C. Formation of Decrypted Text using XOR Operation

Step 1: XOR operation is done between the array CT[] and array DV[] and resultant values are stored in array DT[]. ASCII code is generated from the bit values of array

DT[]. Decrypted characters are generated from the ASCII value and stored in the decrypted text file.

6.2.3. Implementations with Experimental Results

As the formation of ‘0’s and ‘1’s of a single character from a plain text file determines the private key value, so different key is applied for each character of the plain text file. Operators *, + and – are used for even value, odd value and base operator respectively. Value 100 and 10000 is considered for N^{th} term for the prime number and base value respectively. Encryption is done by the secret value derived from the private key and it takes 10047 milliseconds using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM. Table 6.1 represents encryption results.

Table 6.1: Content of Plain Text, Cipher Text and Decrypted Text File for MOEO Scheme

Content of Plain Text File	**Content of Cipher Text**	**Content of Decrypted Text File**
Rgthfdtghhjmnbvcxzasdfghjklpoiuytrewq1245667890-;l;/.,'[[	phyzDLF¾?æ-/?*Ö?Ï²?$üÛÑÕÔïþÎÛÜý¥ßÄ?·¿Ñã-?³?ºíàóýùÞ?ë¸§á??ÈëßÕ°	Rgthfdtghhjmnbvcxzasdfghjklpoiuytrewq1245667890-;l;/.,'[[

Table 6.2 represents the execution results of MOEO scheme for different types of files using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM.

Table 6.2: Representation of Execution Results of MOEO Scheme on Different Types of Files

Name of Plain Text File	Plain Text File Size (Byte)	Encrypted File Size(Byte)	Time needed for Encryption (Milliseconds)	Time needed for Decryption (Milliseconds)
loadfix.com	1131	1131	120031	120012
ReadMe.txt	286	286	37734	37717
WINSTUB.EXE	578	578	82968	82925
VIAPCI.SYS	2712	2712	83235	83213
iconlib.dll	2560	2560	74312	74295
README.COM	4217	4217	390141	390124
LICENSE.TXT	4829	4829	120360	120332
mqsvc.exe	4608	4608	113407	113389
rootmdm.sys	5888	5888	148156	148124
KBDAL.DLL	6656	6656	161329	161315
diskcomp.com	9216	9216	906187	906153
TechNote.txt	9232	9232	244281	244262
label.exe	9728	9728	209969	209947
sffp_mmc.sys	10240	10240	222594	222573
panmap.dll	10240	10240	266375	266359

Figure 6.3.graphically represents how encryption time change depends on file size. As encryption is done on bit level so file type does not make any impact on encryption time.

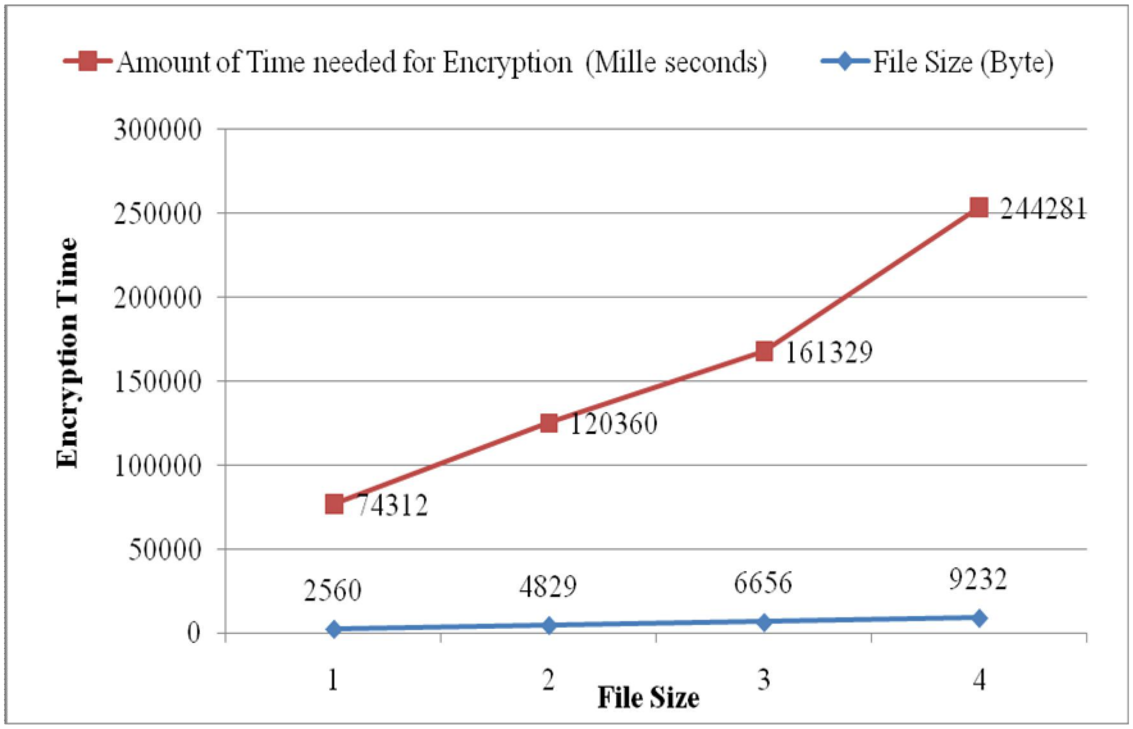


Figure 6.3: Representation of Relationship between File Size and Encryption Time

6.2.4. Security analysis of Multiple Operator and Even Odd position based text encryption scheme (MOEO)

Table 6.3 represents chi-square and degree of freedom values of the implemented MOEO scheme for different types of files where chi-square value and degree of freedom value are calculated as per the equation 3.1 and 3.2 respectively mentioned in chapter 3.

Table 6.3: Representation of Chi-Square and Degree of Freedom Values generated by MOEO Scheme on Different Files

File Name	**File size (Byte)**	**Encryption Time (Milliseconds)**	**Multiple Operator and Even Odd position based text encryption scheme (MOEO)**	
			Chi-Square Value	**Degree of Freedom**
loadfix.com	1131	120031	8216.072266	228
README.COM	4217	390141	70892.250000	250
diskcomp.com	9216	906187	210041.203125	251
ReadMe.txt	286	37734	240.151215	153
LICENSE.TXT	4829	120360	6728.769043	223
TechNote.txt	9232	244281	12392.061523	223
WINSTUB.EXE	578	82968	620.333374	70
mqsvc.exe	4608	113407	2918791.750000	223
label.exe	9728	209969	2807712.000000	249
VIAPCI.SYS	2712	83235	2712.000000	190
rootmdm.sys	5888	148156	5888.000000	231
sffp_mmc.sys	10240	222594	1753322.625000	238
iconlib.dll	2560	74312	1959282.875000	144
KBDAL.DLL	6656	161329	368832.031250	241
panmap.dll	10240	266375	10600647.000000	248

Figure 6.4 and Figure 6.5 show the graphical representation of chi-square values and degree of freedom values generated by MOEO scheme on different types of files respectively.

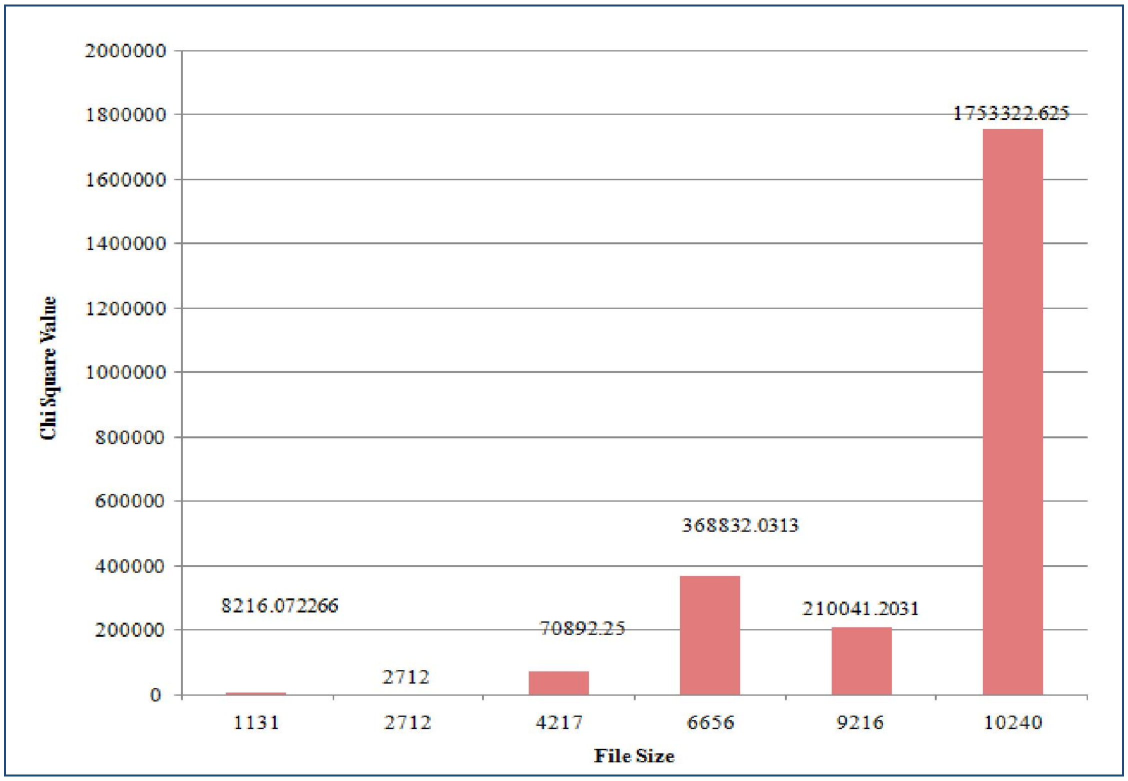


Figure 6.4: Representation of Chi-Square Values generated by MOEO Scheme on Different Files

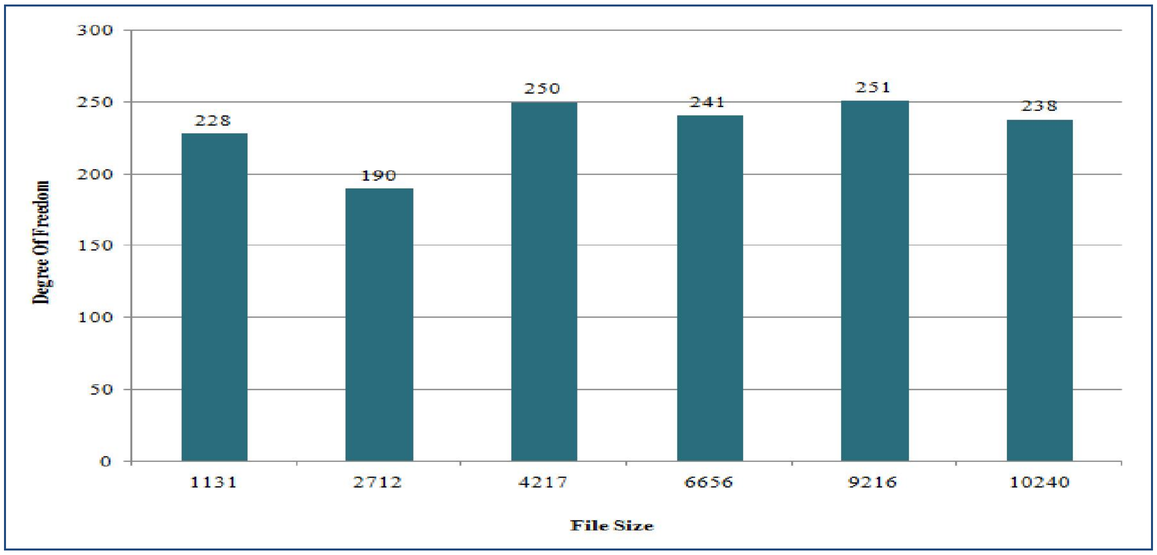
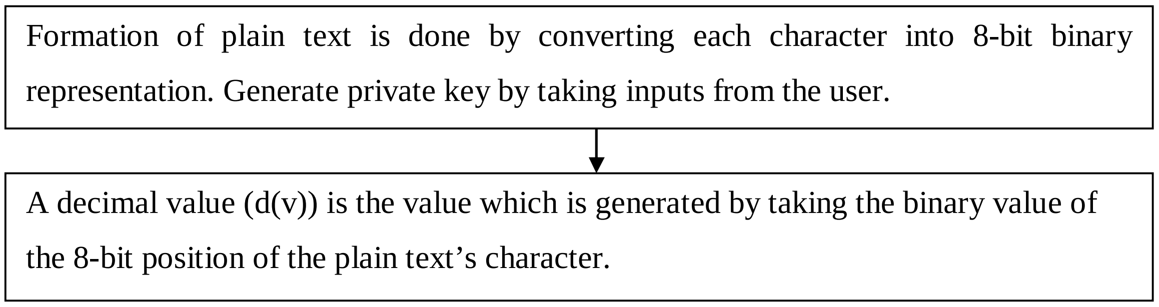


Figure 6.5: Representation of Degree of Freedom Values generated by MOEO Scheme on Different Files

MOEO scheme shows satisfactory performances in respect of the degree of freedom value and chi-square value where security is enhanced by applying large key size.

6.3. Multiple Operator and ASCII Value based text encryption scheme (MOAV)

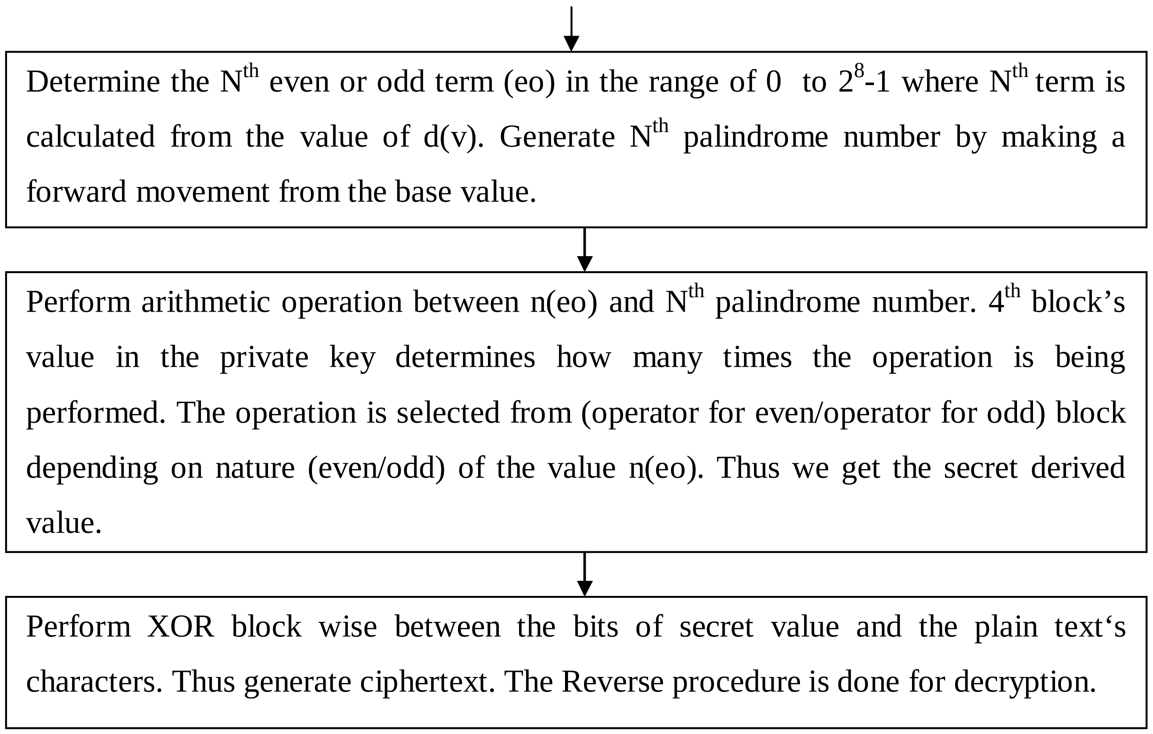
In MOAV[2] scheme, the focus is imposed to design secret procedure which derives separate secret value applied for encryption of each of the plain text's character based on the character's ASCII value. Encryption is carried out using the secret value rather than using the private key value directly. Generation of secret value is done by performing an arithmetic operation between derived decimal value and Nth palindrome number (where N is a positive integer). That decimal value is generated from the binary values of eight-bit representation of a plain text's character. The operators are user defined and are fetched from the respective block of the primary key. Different secret value is generated and applied for encryption of each of the plain text's character as the ASCII code is different for each distinct character of the plain text. Thus the security is enhanced. Figure 6.6 represents the overall procedure for Multiple Operator and ASCII Value based text encryption scheme (MOAV).



Formation of plain text is done by converting each character into 8-bit binary representation. Generate private key by taking inputs from the user.

A decimal value (d(v)) is the value which is generated by taking the binary value of the 8-bit position of the plain text's character.

[2] Published in **International Journal of Engineering Research & Technology (IJERT),** Volume 2, Issue 4, pp. 1433-1438,with title An Approach of Private Key Encryption Technique based on Multiple Operators and Nth Even or Odd Term for ASCII Value of a Plain Text's Character



Determine the Nth even or odd term (eo) in the range of 0 to 2^8-1 where Nth term is calculated from the value of d(v). Generate Nth palindrome number by making a forward movement from the base value.

Perform arithmetic operation between n(eo) and Nth palindrome number. 4th block's value in the private key determines how many times the operation is being performed. The operation is selected from (operator for even/operator for odd) block depending on nature (even/odd) of the value n(eo). Thus we get the secret derived value.

Perform XOR block wise between the bits of secret value and the plain text‘s characters. Thus generate ciphertext. The Reverse procedure is done for decryption.

Figure 6.6: Overall Procedure for Multiple Operator and ASCII Value based text encryption scheme (MOAV)

Section 6.3.1 and section 6.3.2 represents encryption and decryption process respectively. Experimental results and security analysis of MOAV scheme are described in section 6.3.3 and 6.3.4 respectively.

6.3.1. Encryption Process

A. Formation of Plain Text

Step 1: Convert each character of the plain text file into 8-bit binary representation and store the value into an array name PT[]. Carry out the same operations until all the characters of the plain text file are visited.

B. Generation of Private Key

Step 1: Read the inputs for the first, second, fourth, sixth and seventh block of the primary key. Value of third and fifth block is calculated from each character of the plain text file.

Step 2: Convert the input and derived values into bits corresponding to their respective block of the primary key and store the values into an array named KEY[].

The primary key has seven numbers of blocks and its size is 256 bits. First and second block hold the operators applied for the values derived from even or odd bit position's value of a plain text character respectively. Value of the third block defines that the fifth block's value is even or odd. Fourth block's value determines how many times the operator for even or odd is executed. The fifth block represents a value which is generated from bit value of plain text's character in respect of Nth (N is a positive integer with a range of 0 to 2^8-1) even or odd term. The sixth block holds the Nth (N is a positive integer with a range of 0 to 2^{78}-1) term for palindrome number. The seventh block holds the base value. Figure 6.7 represents the key structure of the primary key.

1st block	2nd block	3rd block	4th block	5th block	6th block	7th block
Operator applied for even value o(ep)	Operator applied for odd value o(op)	Selection code for even or odd	Number of time o(op) or o(ep) is to be executed	Nth even/odd term for the bit value of plain text's character n(eo)	Nth term for palindrome number	Base Value
8 bits	8 bits	2 bits	2 bits	8 bits	78 bits	150 bits

Figure 6.7: Structure of 256 bits Private Key

C. Formation of Secret Value for Encryption

Step 1: Decimal value d(v) is generated from the bit values of a plain text's character. The value of d(v) is represented in respect of Nth even or odd term n(eo) in the range of 0 to 2^8-1. Nth palindrome number is calculated from the user mentioned base value with a forward movement.

Step 2: Arithmetic operation is done between n(eo) and Nth palindrome number where the value of the fourth block of primary key defines that the number of times operation is done. The operation is defined by the operators (operator for even/operator for odd) which is selected based on nature (even/odd) of the value n(eo). Thus secret value is generated.

Step 3: Secret value is stored into array called DV[] in respect of binary representation.

D. Ciphertext Formation using XOR Operation

Step 1: XOR operation is done in between array PT[] and array DV[] and store the value in an array named EN[].

Step 2: Generate encrypted character from the ASCII value where the ASCII value is formatted from the value of the array EN[]. Encrypted characters are stored in the ciphertext file.

6.3.2. Decryption Process

A. Conversion of Cipher Text

Each character of ciphertext file is converted into 8-bit binary representation and stored into an array named CT[].

B. Formation of Secret Value for Decryption

Generate the secret value from the key using section 6.3.1.C. Convert the value into binary representation and store the value into an array named DV[].

C. Decrypted Text Formation using XOR Operation

Step 1: XOR operation is done between the array CT[] and array DV[] and the result is stored in array DT[].

Step 2: Final decrypted character is generated from the ASCII value. ASCII value is generated from the value of DT[]. Decrypted characters are stored in the decrypted text file.

6.3.3 Experimental Results and Discussions

As ASCII code value of each character from the plain text file and other inputted operators determine private key value, so separate key is applied for encrypting each character of the plain text file. Operators + and - are applied for even value and odd value respectively. Value 100 and 10000 is considered for N^{th} term for palindrome number and base value respectively. 2 is considered as the value of no. of time operation is performed.

16328 milliseconds are needed for encryption using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM and encryption is carried out by the secret value derived from the private key. Table 6.4 represents encryption results.

Table 6.4: Content of Cipher Text, Plain Text and Decrypted Text File for MOAV Scheme

Plain Text File's Content	Cipher Text File's Content	Decrypted Text File's Content
1234567890qweryyASFGJK LZCBN[];',//!@#$%^&*()_+ }{":<>?	;9?v=z3{}ossKYM-A>IQ W1-%%+)/6-#U!wq -~zv5-	1234567890qweryyASFGJKL ZCBN[];',//!@#$%^&*()_+}{" :<>?

Table 6.5 shows the results of executions of MOAV scheme for different file types using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM.

Table 6.5: Representation of Encryptions Results of MOAV Scheme on Different Files

File Name (Plain Text)	Size of Plain Text File (Byte)	Size of Encrypted File(Byte)	Encryption Time (Milliseconds)	Decryption Time (Milliseconds)
loadfix.com	1131	1131	31703	31652
ReadMe.txt	286	286	204188	204127
WINSTUB.EXE	578	578	46125	46078
VIAPCI.SYS	2712	2712	128125	128075
iconlib.dll	2560	2560	164141	164111
README.COM	4217	4217	93312	93273
LICENSE.TXT	4829	4829	168640	168615
mqsvc.exe	4608	4608	75000	74972
rootmdm.sys	5888	5888	210265	210221
KBDAL.DLL	6656	6656	203125	203113
diskcomp.com	9216	9216	313484	313457
TechNote.txt	9232	9232	314844	314812
label.exe	9728	9728	326172	326136
sffp_mmc.sys	10240	10240	354766	354719
panmap.dll	10240	10240	413844	413811

Figure 6.8 graphically represents how file size makes an impact on encryption time. Encryption time increases as the size of the file are increased.

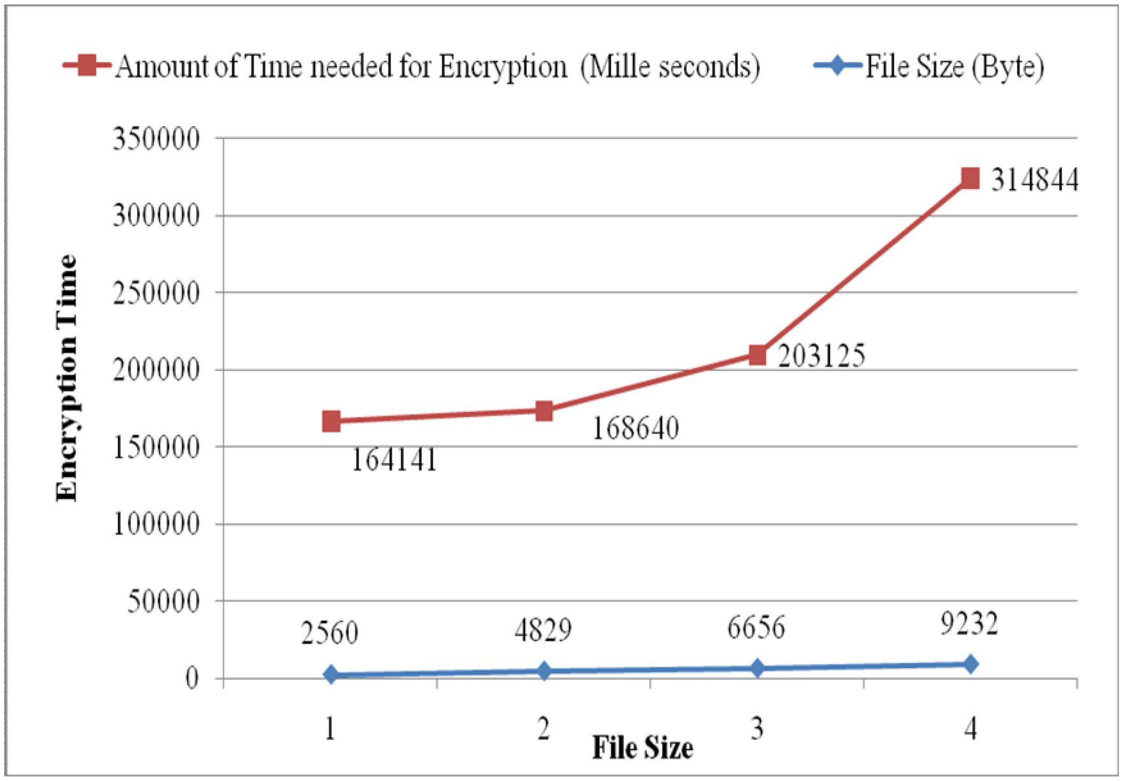


Figure 6.8: Representation of Impact of File Size over Encryption Time

6.3.4. Security Analysis of Multiple Operator and ASCII Value based text encryption scheme (MOAV)

Table 6.6 shows the degree of freedom and chi-square values of MOAV scheme on different types of files where chi-square value and degree of freedom value are calculated as per the equation 3.1 and 3.2 respectively mentioned in chapter 3.

Table 6.6: Resultant Chi-Square and Degree of Freedom Values from MOAV Scheme

File Name (Plain Text)	**File Size (Byte)**	**Encrypted File Size (Byte)**	**Multiple Operator and ASCII Value based text encryption scheme (MOAV)**	
			Chi-Square Value	**Degree of Freedom**
loadfix.com	1131	1131	68012.914063	175
README.COM	4217	4217	191009.515625	245
diskcomp.com	9216	9216	247490.781250	248
ReadMe.txt	286	286	5488.802734	57
LICENSE.TXT	4829	4829	49900.523438	118
TechNote.txt	9232	9232	48316.464844	125
WINSTUB.EXE	578	578	578	75
mqsvc.exe	4608	4608	1085233.000000	228
label.exe	9728	9728	349128.875000	248
VIAPCI.SYS	2712	2712	110543.289063	212
rootmdm.sys	5888	5888	621606.812500	240
sffp_mmc.sys	10240	10240	569319.312500	252
iconlib.dll	2560	2560	9661.562500	150
KBDAL.DLL	6656	6656	3065705.250000	234
panmap.dll	10240	10240	1077958.875000	247

Figure 6.9 and Figure 6.10 show graphical representation of resultant chi-square and degree of freedom values from MOAV scheme on different file types respectively.

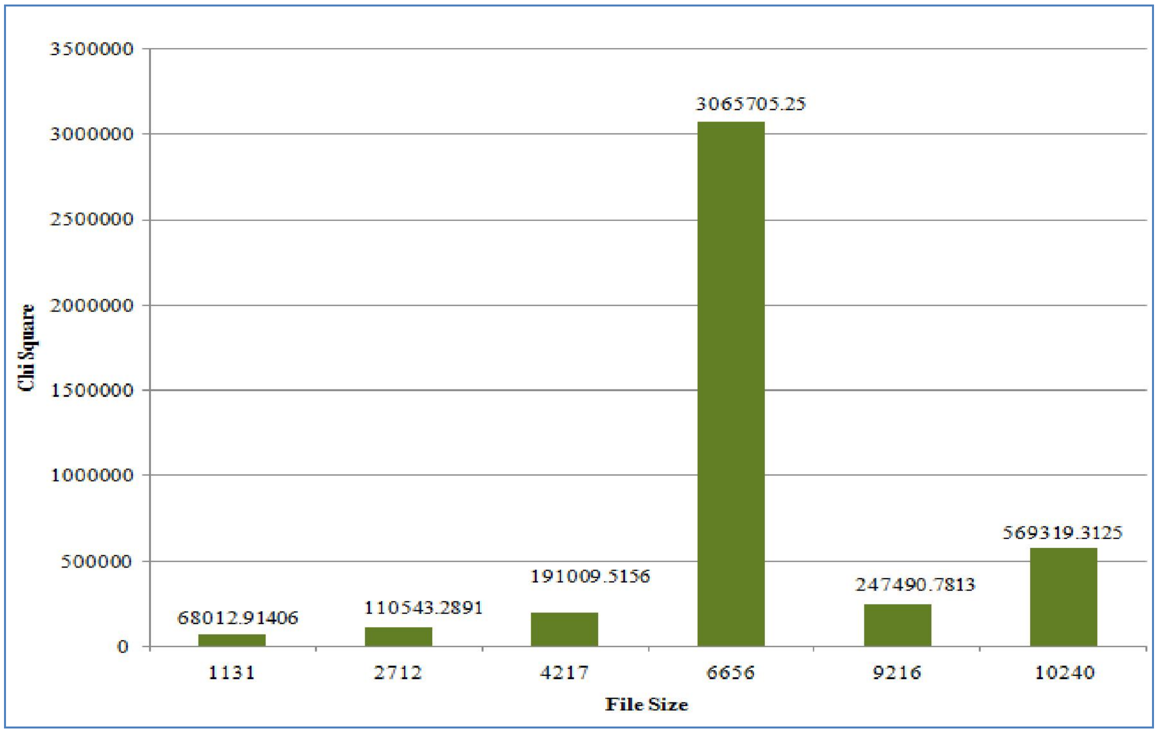


Figure 6.9: Representation of Resultant Chi-Square Values from MOAV Scheme

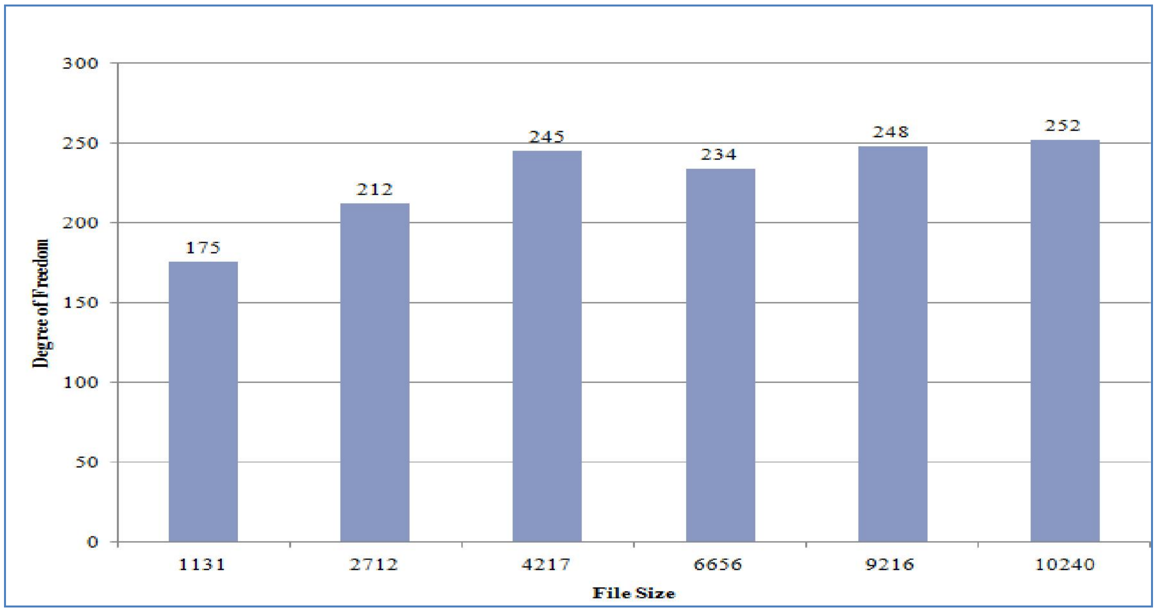


Figure 6.10: Representation of Degree of Freedom Values generated from MOAV Scheme

Large key size increases the security in great extent for MOAV scheme. Besides this, satisfactory performance is measured in respect of Chi-Square value and degree of freedom value.

6.4. Multiple Operator and number of Zeros and Ones based text encryption scheme (MOZO)

In MOZO[3] scheme both the encryption and decryption are carried out by applying the secret value derived from private key based on the number of ‘0’ and ‘1’ present in bit representation of a plain text’s character. Generation of secret value is carried out by performing arithmetic operation between Nth amicable number (where N is a positive integer supplied by user) and derived decimal value where decimal value is generated by executing arithmetic operation between numbers of ‘0’s and numbers of ‘1’s counted from a plain text character’s eight bit binary representation. Base value and arithmetic operators are supplied from the user and are fetched from respective blocks of the primary key. Different secret value is generated and used for encryption of each of the plain text’s character as the count of ‘0’s and ‘1’s are different for several numbers of characters present in the plain text file. Thus this scheme enhances the security. Figure 6.11 represents the overall procedure for Multiple Operator and number of Zeros and Ones based text encryption scheme (MOZO).

Formation of plain text is done by converting each character into 8-bit binary representation. Generate private key by inputs from the user.

↓

[3] Published in **International Journal of Innovative Technology and Exploring Engineering (IJITEE),** Volume 2, Issue 6, pp. 94-98, with title An approach of Bitwise Private-key Encryption Technique based on Multiple Operators and Numbers of 0 and 1 counted from Binary Representation of Plain Text‘s Single Character

One intermediate result (IR) is calculated first. IR = [N (1)-N (0)] o(N) N(1)(where N(1) and N(0) are the counts of ‘1’ and ‘0’ from a plain text’s character respectively, o(N) is Operator between N(0) and N(1)). The N^{th} amicable number is generated with a forward movement from the base value (where N is a positive integer with a range of 0 to 2^{82}-1). Then secret value (SV) is calculated. SV= (N^{th} amicable number) o(B)[Number of times o(B) is executed] IR (where o(B) is base operator).

Perform XOR block wise between the bits of secret value and the plain text’s characters. Thus generate cipher text. The reverse procedure is followed for decryption.

Figure 6.11: Overall procedure for Multiple Operator and number of Zeros and Ones based text encryption scheme (MOZO)

Section 6.4.1 and section 6.4.2 represent the encryption and decryption process. Experiment results are discussed in section 6.4.3 and section 6.4.4 represents the security analysis of MOZO scheme.

6.4.1. Encryption Process

A. Formation of Plain Text

Step 1: Convert each character of plaintext file into 8-bit binary representation and store the value into an array called PT[].

B. Private Key Formation

Step 1: Read the inputs for the first, second, fifth, sixth and seventh block of the private key. The calculation is carried out from plain text to generate values for the third and fourth block of the private key.

Step 2: Convert the inputs into binary representation corresponding to their respective block's size of the primary key. Store the values into an array named KEY[].

There are seven numbers of blocks in the private key where the length is 256 bits. The first block represents the operator applied for base value. Value of the second block defines how many times the operator is going to be performed. Third and fourth block represents the count of '0' and '1' from eight-bit binary representation of a plain text's character respectively. The fifth block represents the operator applied to the values represented by the fourth and third block. The sixth block holds base value. Seventh blocks stores the N^th^ term for amicable number. Figure 6.12 represents the structure of the private key.

1st block	2nd block	3rd block
Operator applied for base value o(b)	Number of time(s) o(b) is going to be executed	Count of '0' from binary representation of a plain text's character N(0)
8 bits	2 bits	3 bits

4th block	5th block	6th block	7th block
Count of '1' from binary representation of a plain text's character N(1)	Operator o(N) between N(0) and N(1)	Base value	Nth term for amicable number
3 bits	8 bits	150 bits	82 bits

Figure 6.12: Structure of 256 bits Private Key

C. Generation of Secret Value for Encryption

Step 1: Two decimal values N(1) and N(0) are calculated where they represent the count of '1' and '0' from eight-bit binary representation of a plain text's character respectively.

Step 2: Intermediate result (IR) is calculated using the following formula:

IR = [N(1)-N(0)] o(N) N(1)(where N(1) and N(0) are the counts of '1' and '0' from binary representation of a plain text's character respectively, o(N) is operator between N(0) and N(1)).

The Nth amicable number is generated with a forward movement from the base value (where N is a positive integer with a range of 0 to 2^{82}-1).
Then secret value (SV) is calculated using the following manner:

```
For ( int i=1 to NT)
{ SV= (N^th amicable number) o(b) IR }( where o(b) is the base operator and NT=Number of times o(b) is performed).
```

Secret value is converted into binary representation and stored into array named DV[].

D. Formation of Cipher Text using XOR Operation

Step 1: XOR operation is done between array PT[] and array DV[] and the result is stored in array CT[]. ASCII code is generated from the value of array CT[] and the encrypted characters are generated from ASCII value. Thus generate ciphertext file.

6.4.2. Decryption Process

A. Formation of Cipher Text

Convert each character from ciphertext file into 8-bit binary representation and store the value into an array called CT[].

B. Generation of Secret Value for Decryption

Secret value is generated using section 6.4.1.C and the value is converted into binary representation and stored into an array called DV[].

C. Generation of Cipher Text using XOR Operation

Step 1: Perform XOR operation between array CT[] and array DV[] and the result is stored in array DT[]. ASCII value is generated from array DT[] and from there decrypted character is generated. Thus generate the decrypted text file.

6.4.3. Experimental Results and Discussions

As the count of zeros and ones in each character of the plain text file and other inputted operators determine the private key for each character, so different keys are used for encrypting each character. Operators + and - are applied as base operator and operator between (number of zeros) and (number of ones) respectively. Value 10000 and 100 are considered as the base value and N^{th} term for the amicable number respectively. 2 is considered as the value of number of time(s) base operation is performed.

Encryption is carried out by the secret value derived from the private key and 14954 milliseconds are needed for execution using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM. Table 6.7 represents encryption results.

Table 6.7: Content of Plain Text, Decrypted Text and Cipher Text File for MOZO Scheme

Content of Plain Text	Content of Encrypted Text	Content of Decrypted Text
12345678tfeeygewhfwej BHGDFRJHKMH=][';/., @#$%^&*()()_+|{}":?><	=>!8'$'4ftwwiwwydtywx44LF UJJ^FFY_EF?-MK5+?< T/*)N*&&%&%Q9lus,(1..	12345678tfeeygewhfwej BHGDFRJHKMH=][';/., @#$%^&*()()_+|{}":?><

Table 6.8 shows execution results of MOZO scheme for different types of files using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM.

Table 6.8: Representation of Results of Encryptions for MOZO Scheme on Different Files

Plain Text File Name	File Size (Byte)	Size of Encrypted File (Byte)	Time needed for Encryption (Milliseconds)	Time needed for Decryption (Milliseconds)
loadfix.com	1131	1131	113547	113532
ReadMe.txt	286	286	113906	113892
WINSTUB.EXE	578	578	62359	62343
VIAPCI.SYS	2712	2712	148297	148281
iconlib.dll	2560	2560	130156	130145
README.COM	4217	4217	224047	224031
LICENSE.TXT	4829	4829	323500	323479
mqsvc.exe	4608	4608	213578	213557
rootmdm.sys	5888	5888	273968	273953
KBDAL.DLL	6656	6656	286015	285098
diskcomp.com	9216	9216	391781	391759
TechNote.txt	9232	9232	579831	579817
label.exe	9728	9728	410656	410644
sffp_mmc.sys	10240	10240	480703	480687
panmap.dll	10240	10240	592031	592012

Figure 6.13 graphically represents the relationship between file size and encryption time. The graph shows there is a proportional relationship between file size and encryption time.

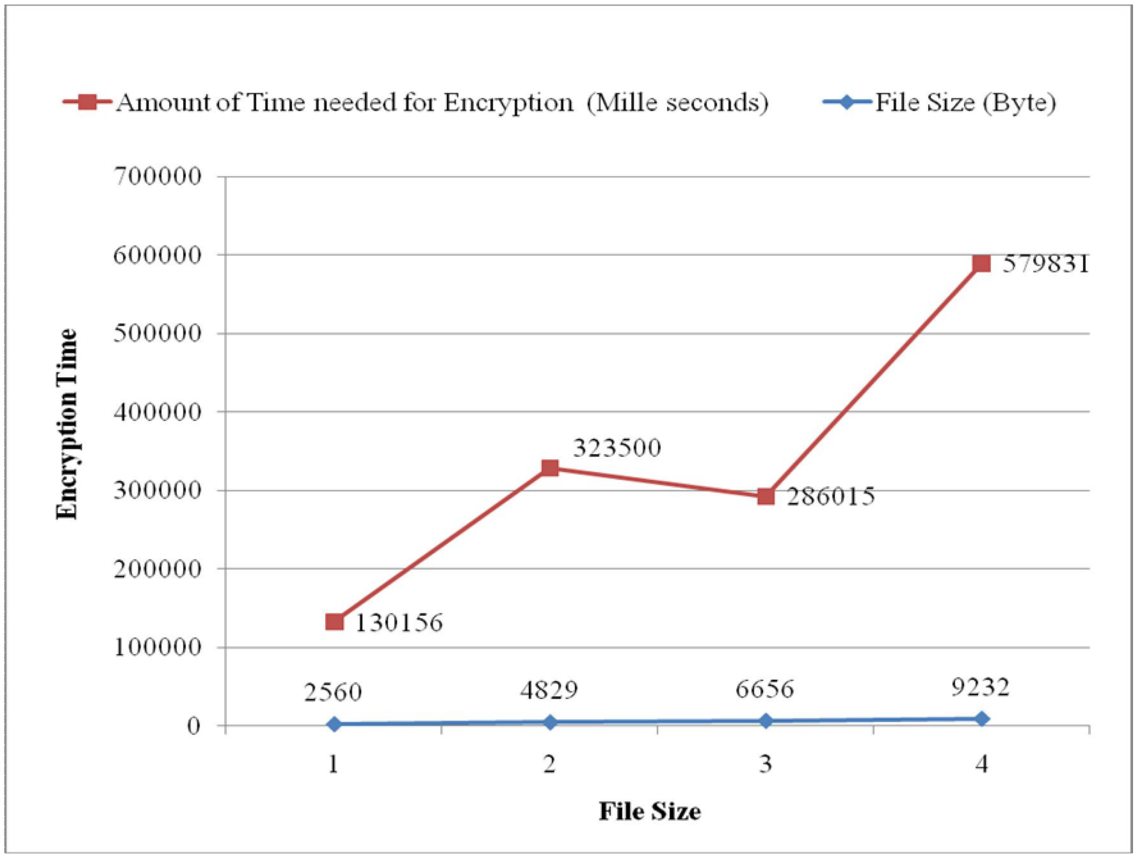


Figure 6.13: Representation of Relationship between Encryption Time and File Size

6.4.4. Security Analysis of Multiple Operator and number of Zeros and Ones based text encryption scheme (MOZO)

Table 6.9 represents Chi-Square values and degree of freedom values of MOZO scheme on different file types where Chi-Square value and degree of freedom value are calculated as per the equation 3.1 and 3.2 respectively mentioned in chapter 3.

Table 6.9: Resultant Degree of Freedom and Chi-Square Values from MOZO Scheme

File Name	**File Size (Byte)**	**Encrypted File Size (Byte)**	**Multiple Operator and number of Zeros and Ones based text encryption scheme (MOZO)**	
			Chi-Square Value	**Degree of Freedom**
iconlib.dll	2560	2560	1959282.875000	144
KBDAL.DLL	6656	6656	7593299.000000	234
panmap.dll	10240	10240	10600647.000000	248
VIAPCI.SYS	2712	2712	74822.421875	211
rootmdm.sys	5888	5888	1753322.625000	238
sffp_mmc.sys	10240	10240	3482557.250000	252
WINSTUB.EXE	578	578	620.333374	70
mqsvc.exe	4608	4608	2918791.750000	223
label.exe	9728	9728	2807712.000000	249
ReadMe.txt	286	286	607.052612	58
LICENSE.TXT	4829	4829	22186.935547	118
TechNote.txt	9232	9232	112907.921875	129
loadfix.com	1131	1131	14604.628906	181
README.COM	4217	4217	147652.125000	247
diskcomp.com	9216	9216	1782321.875000	247

Figure 6.14 and Figure 6.15 represent a graphical representation of the degree of freedom values and Chi-Square values from MOZO scheme for different types of file.

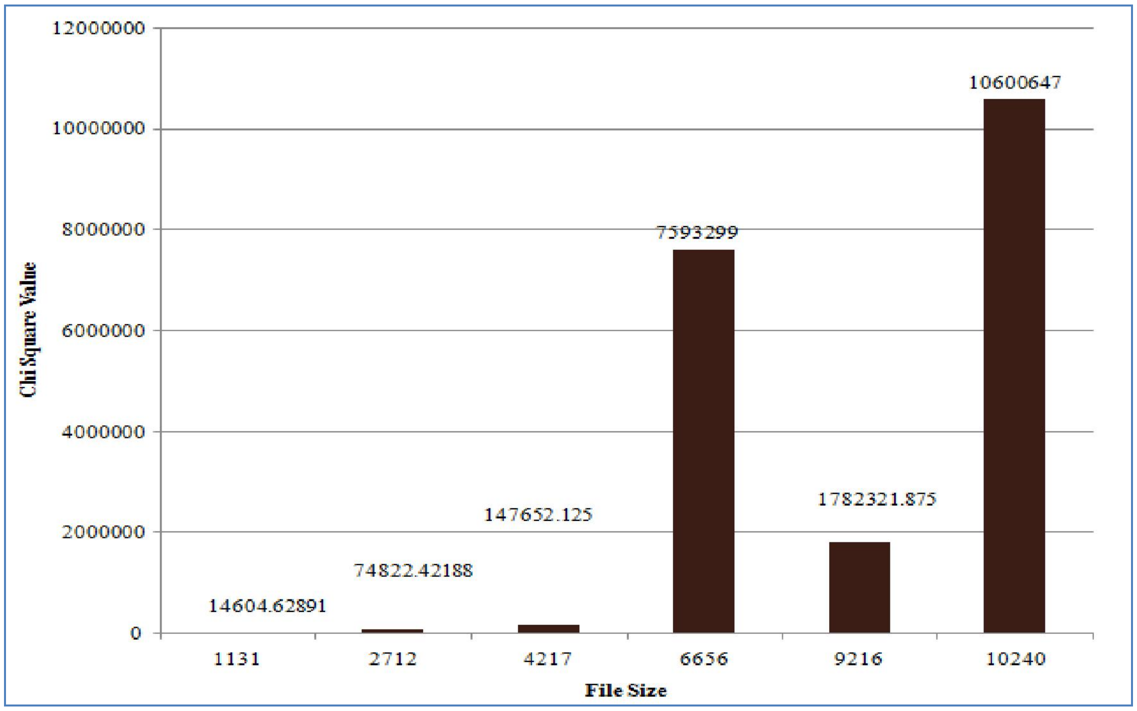


Figure 6.14: Representation of Chi-Square Values generated from MOZO Scheme

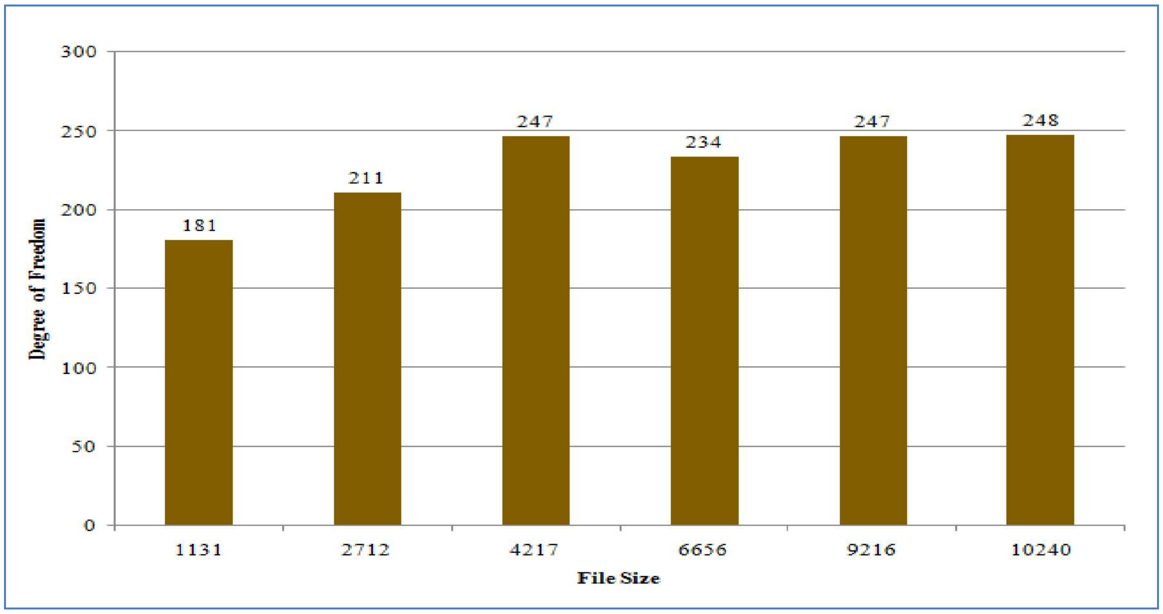


Figure 6.15: Representation of Degree of Freedom Values generated from MOZO Scheme

Satisfactory Chi-Square value and degree of freedom value are measured for MOZO scheme where the presence of higher key length has increased the security to great extent.

6.5. Conclusion

A separate secret value is generated and applied for encryption of each character of plain text by the implemented schemes as formation of ‘0’ and ‘1’ of each distinct character, ASCII code for each distinct character and number of ‘0’s, ’1’s present in several characters from the plain text file are different. So the security is increased.

As the secret value formation procedure is private and distributed between only sender and receiver, so the system is secured in spite of stealing of the private key. Thus the security is enhanced.

Encryption time is independent of file types as the encryption is carried out in bit level for the implemented schemes. So conclusions may be carried out that the newly developed text encryption schemes may provide great security for text encryption.