Chapter 5

Text Encryption Building Blocks using Special Numbers with Alphabetic Group or Cipher Sequencing

5.1. Overview

Private key encryption scheme uses the same cryptographic key both for encryption and decryption where the key may be totally identical or simple transformation may be present between the keys. Different existing text encryption schemes are discussed in section 2.5 of chapter 2 of this thesis. The distribution of private key through open communication channel may reduce the security level as there is a high chance of interpretation of the key. So the focus has been imposed to design secret procedures which retrieve the secret value based on the private key and applied it for encryption rather than using the direct key value. Some new text encryption schemes are developed based on special numbers, alphabetic group, operators and cipher sequencing and their superiority are also being compared with existing standard algorithms.

This chapter contains four text based encryption schemes. They are Prime number with Alphabetic Group based text encryption scheme (PAG)[1], Palindrome number with Alphabetic Group and Operator based text encryption scheme (PAGO)[2], Armstrong and Perfect number with Cipher Sequencing based text encryption scheme (APCS)[3] and Amicable number with Cipher Sequencing based text encryption scheme (ACS)[4]. Performances of the implemented schemes are measured in respect of standard parameters like execution time, chi-square value and degree of freedom.

In this chapter, Prime number with Alphabetic Group based text encryption scheme (PAG) in section 5.2, Palindrome number with Alphabetic Group and Operator based text encryption scheme (PAGO) in section 5.3, Armstrong and Perfect number with Cipher Sequencing based text encryption scheme (APCS) in section 5.4 and Amicable number with Cipher Sequencing based text encryption scheme (ACS) in section 5.5 have been discussed. A conclusion has been drawn in section 5.6.

5.2. Prime number with Alphabetic Group based text encryption scheme (PAG)

In PAG[1] approach, both encryption and decryption are carried out by the secret value derived from the private key. Secret value is generated by taking the combination of N^{th} Consonant, N^{th} Vowel, N^{th} Semivowel and N^{th} Prime number. Where N^{th} term is user defined and N is a positive integer. Value of the N^{th} term of the consonant, vowel, semivowel and prime number is calculated in respect of user defined base value towards its backward or forward direction. A user defined sequence number is applied at the time of combining all these N^{th} terms. Thus the secret value is generated. Generation of secret value is governed by the content of base value, alphabetic group, N^{th} term, and combination sequence. So as these values are changed that generates a new secret value which is applied for encryption. Thus enhance security. As compared with AES, Twofish schemes, implemented schemes provide a great result in respect of chi-square value and degree of freedom value. Figure 5.1 represents the overall procedure for PAG scheme.

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

↓

The values of the 2^{nd} block, 3^{rd} block and 4^{th} block of private key are used to determine the N^{th} term of vowel, semi-vowel, and consonant respectively.

↓

Searching of N^{th} prime is done by making a forward (1) or backward (0) movement from user defined base value where N is a positive integer.

[1] Published in **International Journal of Innovative Technology & Adaptive Management (IJITAM),** Volume 1, Issue 7, pp. 9-14, with title An Approach of Bit-level Private-key Encryption Scheme based on Alphabetic group and Prime Number In Selective Mode

The user defined sequence value stored in 1^{st} block of private key is used to combine the results (N^{th} (vowel, semi-vowel, consonant and prime number)) together and secret value is generated.

Perform XOR operation block wise between the bits of secret value and the plain text‘s characters. Thus generate ciphertext.

Figure 5.1: Overall Procedure for Prime number with Alphabetic Group based text encryption scheme (PAG)

Section 5.2.1 and section 5.2.2 describes the encryption process and decryption process respectively. Section 5.2.3 represents the experiment results and security analysis of the developed algorithm is represented in section 5.2.4.

5.2.1. Encryption Process

A. Formation of Plain Text

Step 1: Read each character from the inputted file and convert the character into an 8-bit binary representation. The bit-values of each character are stored into an array named PT[]. In this way, all the characters of the inputted file are converted into binary form and plain text is generated.

B. Generation of Private Key

Step 1: Read the user inputs for all the blocks of the private key. Convert the inputs into bits corresponding to their respective blocks of the key and store the value in the array called P_KEY[] of size 256.

Seven numbers of blocks are present in the user define private key where the size of the key is 256 bits. The first block represents the combination sequence value used for combining other derived values from different blocks of the private key. Second, third and fourth blocks represent the N^{th} term for the vowel, semivowel and consonant respectively where N is a positive integer. The fifth block defines the forward and backward movement. The N^{th} term for prime value is defined in the sixth block. User supplied base value is present in the seventh block of the key. Figure 5.2 represents the key structure.

1^{st} block	2^{nd} block	3^{rd}block	4^{th}block	5^{th}block	6^{th}block	7^{th}block
Combination Sequence	N^{th} term for vowel	N^{th} term for semivowel	N^{th} term for consonant	Backward or forward movement	N^{th} term for prime number	Base value
5 bits	3 bits	2 bits	5 bits	2 bits	89 bits	150 bits

Figure 5.2: Structure of 256 bits Private Key

C. Generation of Secret Value from Private Key

Step 1: N^{th} Prime number is generated by making forward or backward movement from user defined base value. N^{th} vowel, semivowel and consonant are also computed as per the inputted value present in the corresponding block of the private key. All the computed vowel, semivowel, consonant and prime number are combined as the combination sequence to generate the secret value applied for encryption and decryption. Secret value is converted into bits and stored into an array called DV[].

D. Formation of Ciphertext using XOR Operation

Step 1: Secret value is represented in binary representation and those bits are grouped into multiple blocks each of 8-bits where numbers of blocks are defined by secret value. XOR operation is carried out between the bits of plain text character (PT[]) and the bits of each block of secret value (DV[]) in a cumulative manner. Resultant bit value of final XOR operation is converted into corresponding ASCII code and thus generate an encrypted character. In this way, all the character of plain text final is encrypted and ciphertext file is generated. Private Key and cipher are shared to the receiver.

5.2.2. Decryption Process

A. Formation of Binary Representation of Ciphertext

Step 1: Read each character from ciphertext file and convert it into 8-bit binary representation. The bit-values of each character are stored in an array named CT[]. Continue the activity for all the characters of ciphertext file.

B. Formation of Secret Value from Primary Key

Formation of secret value is carried out by using section 5.2.1.C.

C. Formation of Decrypted Text using XOR Operation

Step 1: Bit representation of secret value is grouped into multiple blocks each of 8-bits. Cumulative XOR operation is performed between the bits of ciphertext (CT[]) and the bits of each block of secret value (DV[]) and the result is converted into ASCII code from where the decrypted character is generated. In this way, all the character of ciphertext file is decrypted.

5.2.3. Implementation with Experiment Results

Encryption is carried out using the 4^{th} prime number from user defined base value 100 with a forward movement. 18922 milliseconds are needed for encryption using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM. Table 5.1 shows the encryption result.

Table 5.1: Content of Plain Text, Ciphertext and Decrypted Text File

Content of Plain Text File (PT.txt)	Content of Ciphertext File (CT.txt)	Content of Decrypted Text File (DT.txt)
qwertyuioplkjhgfdsazxcbnm	sugpv{wkmrnihjedfqcxza`lo	qwertyuioplkjhgfdsazxcbnm

Table 5.2 shows the execution result of the algorithm on different types of files (.com, .exe, .txt, .dll, .sys) using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM.

Table 5.2: Execution Results of PAG Algorithm for Different File Types

Name of the Plain Text File	Size of Plain Text File (Byte)	Size of Encrypted File (Byte)	Time needed for Encryption (Milliseconds)	Time needed for Decryption (Milliseconds)
loadfix.com	1131	1131	52294	52280
ReadMe.txt	286	286	55975	55930
WINSTUB.EXE	578	578	35883	35817
VIAPCI.SYS	2712	2712	83916	83893
iconlib.dll	2560	2560	88577	88532
README.COM	4217	4217	127921	127897
LICENSE.TXT	4829	4829	155739	155711
mqsvc.exe	4608	4608	87773	87724
rootmdm.sys	5888	5888	115956	115913
KBDAL.DLL	6656	6656	172681	172632
diskcomp.com	9216	9216	218038	218011
TechNote.txt	9232	9232	188882	188833
label.exe	9728	9728	165797	165745
sffp_mmc.sys	10240	10240	226806	226785
panmap.dll	10240	10240	171292	171219
kb16.com	14710	14710	273709	273674

ROMAN.TXT	14423	14423	270968	270918
shadow.exe	4848	4848	156108	156078
smclib.sys	14592	14592	225844	225817
tcpmib.dll	14848	14848	347675	347627

The relationship between file size and encryption time is graphically represented in Figure 5.3. Encryption or decryption time does not depend on the type of the file rather it depends on the size of the file as encryption is done on bit level.

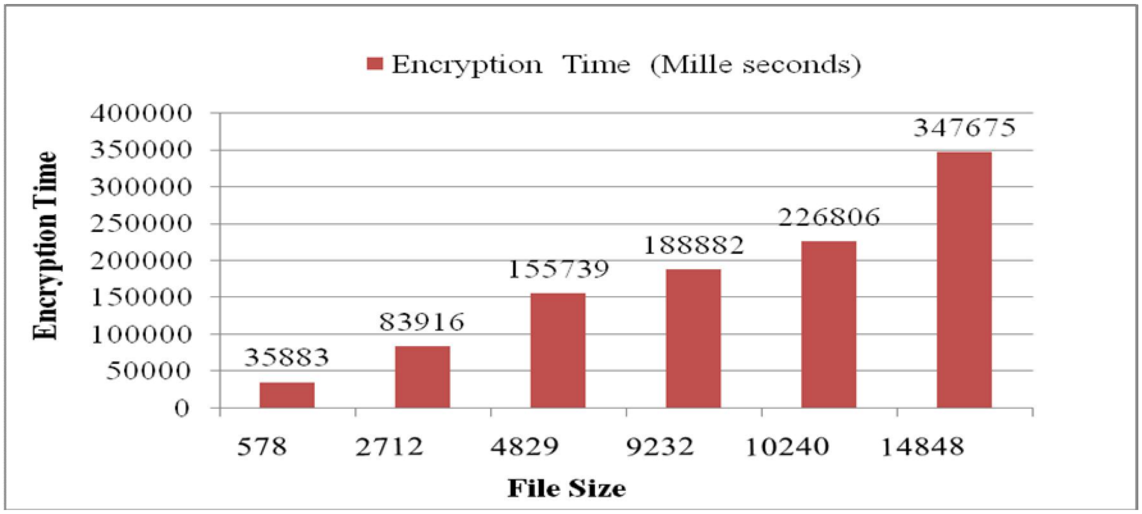

Figure 5.3: Representation of the Relationship between Encryption Time and File Size

5.2.4. Security analysis of Prime number with Alphabetic Group based text encryption scheme (PAG)

Table 5.3 and Table 5.4 shows the comparison between the results of implemented PAG scheme with AES-256 algorithms and Twofish scheme respectively. Chi-square value and degree of freedom value are considered as standard parameters where chi-square value and degree of freedom value are calculated as per equation 3.1 and 3.2 respectively mentioned in chapter 3.

Table 5.3: Comparison between Implemented PAG Algorithm with AES-256 Algorithm in respect of Chi-Square Values and Degree of Freedom Value

File Name	**File Size (Byte)**	**Results for Prime number with Alphabetic Group based text encryption scheme (PAG)**		**Results for AES-256 bit Scheme**	
		Chi-Square Value	**Degree of Freedom**	**Chi-Square Value**	**Degree of Freedom**
loadfix.com	1131	26221.970703	165	1786.443359	254
README.COM	4217	23029.445313	244	10778.030273	241
diskcomp.com	9216	1799318.375000	245	58196.539063	244
LICENSE.TXT	4829	20030.685547	97	6727.230469	255
TechNote.txt	9232	49524.527344	107	12854.843750	255
ROMAN.TXT	14423	169029.953125	209	26362.699219	202
WINSTUB.EXE	578	123078.804688	72	614.405762	242
mqsvc.exe	4608	2944534.500000	235	27169.955078	232
label.exe	9728	1836753.250000	247	73068.726563	245
VIAPCI.SYS	2712	317208.031250	202	7404.197754	255
rootmdm.sys	5888	1516031.750000	234	28955.144531	255
sffp_mmc.sys	10240	1452877.000000	253	60305.925781	250
iconlib.dll	2560	1290453.375000	140	6509.628418	255
KBDAL.DLL	6656	2302158.000000	229	44483.308594	225
panmap.dll	10240	2177440.000000	247	78672.390625	242

Table 5.4: Comparison between Implemented PAG Algorithm with Twofish Algorithm in respect of Chi-Square Values and Degree of Freedom Values

File Name	File size (Byte)	Prime number with Alphabetic Group based text encryption scheme (PAG)		Results for Twofish Scheme	
		Chi-Square Value	Degree of Freedom	Chi-Square Value	Degree of Freedom
loadfix.com	1131	26221.970703	165	3550.732178	247
README.COM	4217	23029.445313	244	10450.395508	243
diskcomp.com	9216	1799318.375000	245	72066.039063	245
LICENSE.TXT	4829	20030.685547	97	6840.962891	255
TechNote.txt	9232	49524.527344	107	13748.231445	255
ROMAN.TXT	14423	169029.953125	209	23264.337891	205
WINSTUB.EXE	578	123078.804688	72	1401.833374	133
mqsvc.exe	4608	2944534.500000	235	92500.148438	255
label.exe	9728	1836753.250000	247	134848.796875	255
VIAPCI.SYS	2712	317208.031250	202	7470.710449	255
rootmdm.sys	5888	1516031.750000	234	31121.910156	255
sffp_mmc.sys	10240	1452877.000000	253	78186.835938	252
iconlib.dll	2560	1290453.375000	140	29127.291016	255
KBDAL.DLL	6656	2302158.000000	229	79729.109375	255
panmap.dll	10240	2177440.000000	247	88304.062500	245

Figure 5.4 and Figure 5.5 show the graphical comparison between the results of implemented PAG scheme with AES-256 algorithm and Twofish scheme in respect of Chi-Square value and degree of freedom value respectively.

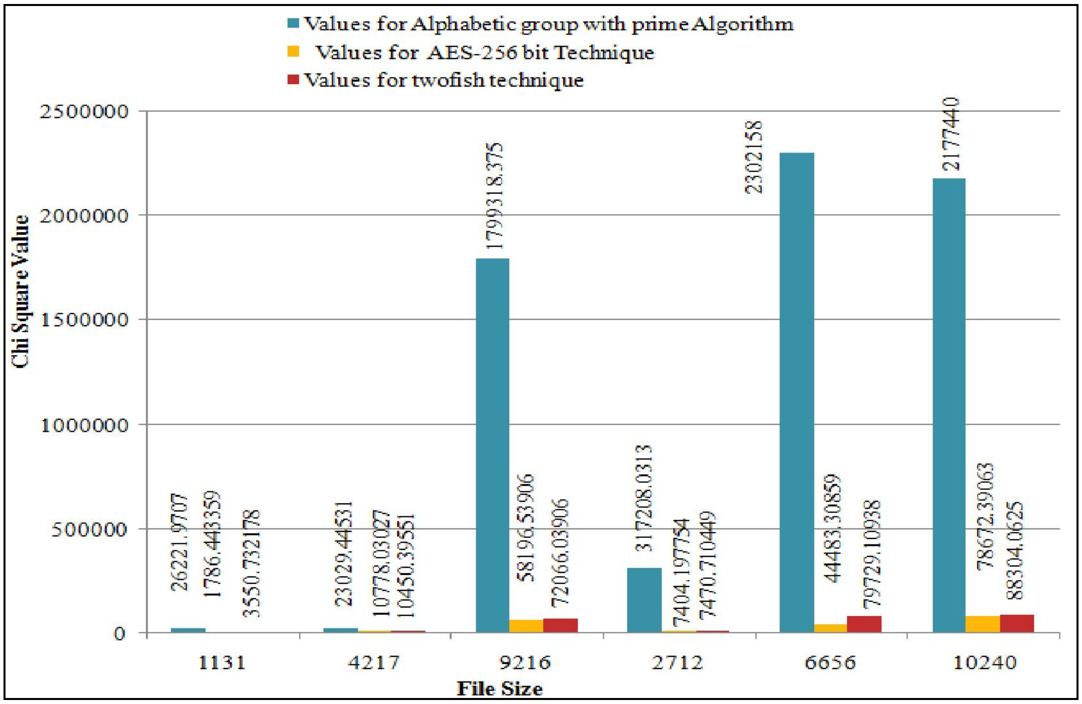

Figure 5.4: Comparison between PAG Algorithm, AES and Twofish Algorithm in respect of Chi-Square value

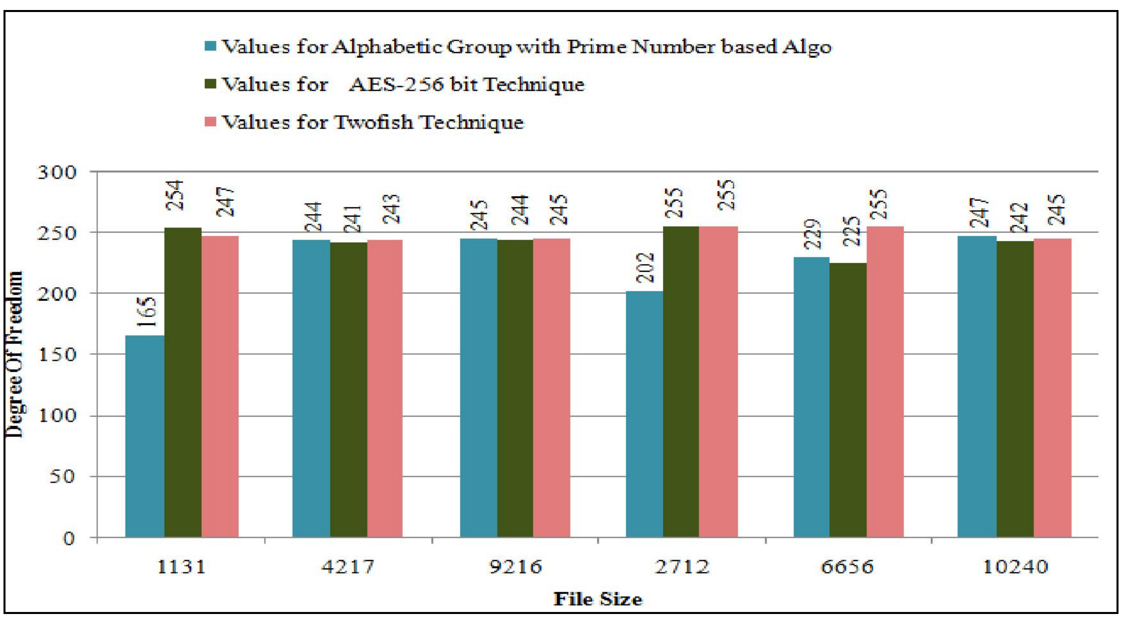

Figure 5.5: Comparison between PAG Algorithm, AES and Twofish Algorithm in respect of Degree of Freedom

The PAG scheme shows satisfactory results in respect of Chi-Square value and degree of freedom value. Extremely large key size provides great security for the implemented scheme.

5.3. Palindrome number with Alphabetic Group and Operator based text encryption scheme (PAGO)

Traditional private key encryption schemes suffer from lack of security as the challenge is to distribute the private key without interpretation through the public communication channel. In this PAGO[2] scheme both the decryption and encryption are carried out by applying the secret value generated from the private key. N^{th} palindrome number is generated from the user defined base value with a forward or backward movement where N is user defined and a positive number. Similarly, N^{th} element from consonant, special character, vowel or semivowel group is also generated where the value of N is user inputted. The intermediate value is calculated by performing user defined operation between base value and N^{th} palindrome number. Lastly, secret value is generated by combining all these values together directed by a user defined sequence mention in the private key. Changing of base value, the N^{th} term, alphabetic group and combination sequence generate different secret value used for encryption. Thus an attempt is made to enhance the security. As compared with AES, Serpent schemes, implemented scheme provides great result in respect of Chi-Square value and degree of freedom. Figure 5.6 represents the overall procedure for Palindrome number with Alphabetic Group and Operator based text encryption scheme (PAGO).

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

[2] Published in **International Journal of Innovative Technology & Adaptive Management (IJITAM),** Volume 1, Issue 7, pp. 21-27, with title An Approach of Bit-level Private-key Encryption Scheme based on Alphabetic Group, User-defined Operator and Palindrome Number in Selective Mode

Determine N^{th} Palindrome Number by making forward or backward movement from user defined base value where N is a positive integer and generate N^{th} consonant, special character, vowel or semivowel.

Generate intermediate value by performing an arithmetic operation between base value and N^{th} palindrome number specified in the 5^{th} block of the private key.

The user defined sequence value stored in the 4^{th} block is being used to combine the results (N^{th} (consonant, special character, vowel and semivowel, intermediate value)) together and generate secret value.

Perform XOR operation 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 5.6: Overall Procedure for Palindrome number with Alphabetic Group and Operator based text encryption scheme (PAGO)

Here section 5.3.1 represents the encryption process. Section 5.3.2 describes the decryption process. Experiment results are represented in section 5.3.3. Section 5.3.4 shows the security analysis of PAGO scheme.

5.3.1. Encryption Process

A. Plain Text Formation

Step 1: Each character from the inputted file is converted into 8-bit binary representation and stored in an array named PT[]. Similar activity is carried out for all the characters in inputted file and plain text is generated.

B. Generation of Private Key

Step 1: Read and convert the inputs into bits corresponding to their respective blocks of the key and store the value in the array called P_KEY[] of size 256.

Private key has 8 numbers of blocks and its size is 256 bits. First, second, third and fourth block of the private key contains N^{th} term for consonant, N^{th} term for special characters, N^{th} term for vowel or semivowel and N^{th} term for combination sequence respectively where N is a positive integer. The fifth block holds user defined operator value. Value of the sixth block defines forward or backward movement. The seventh block holds the value of N^{th} term for palindrome number. Base value defined by the user is stored in block eight. Figure 5.7 represents the structure of the private key.

1st block	2^{nd} block	3^{rd} block	4^{th} block
Value of N^{th} term for consonant selection	Value of N^{th} term for special character selection	Value of N^{th} term for Vowel or semivowel selection	Value of N^{th} term for combination sequence selection
5 bits	5 bits	3 bits	5 bits

5^{th} block	6^{th} block	7^{th} block	8^{th} block
Value of User defined operator	Value to determine Forward or Backward movement	Value of N^{th} term for Palindrome number selection	User defined base value
6 bits	2 bits	80 bits	150 bits

Figure 5.7: Structure of 256 bits Private Key

C. Generation of Secret Value from Primary Key

Step 1: Selection of N^{th} palindrome number is done by making a user defined forward or backward movement from the base value where N is a positive integer. Generate N^{th} consonant, special character, vowel or semivowel as per the values present in the corresponding block of the private key. The intermediate value is generated by performing the arithmetic operation between the base value and N^{th} palindrome value where the inputted operator is fetched from the specific block of the private key.

Step 2: Combination sequence is used to combine N^{th} consonant, special character, vowel or semivowel, and intermediate value together and formulate secret value. Secret value is converted into bits and stored into an array called DV[].

D. Ciphertext Generation using XOR Operation

Step 1: Derived secret value is converted into a binary representation which is grouped into multiple blocks with a size of 8-bits of each block where block size is defined by secret value. XOR operation is performed in between the bits contained in each block of secret value (DV[]) and bits of plain text character (PT[]) in a cumulative manner. Bit value generated after the final XOR operation is converted into ASCII code which generates an encrypted character. Similar activities are carried out for encrypting all the characters from plain text and generate ciphertext file. Ciphertext and private key are shared to the receiver.

5.3.2. Decryption Process

A. Conversion of Ciphertext

Step 1: Read and convert each character from ciphertext file into 8-bit binary representation and store the value into an array called CT[]. Perform a similar activity for all the characters of ciphertext file.

B. Generation of Secret Value from Private Key

Formation of secret value is carried out by using section 5.3.1.C.

C. Formation of Decrypted Text using XOR Operation

Step 1: Binary representation of derived secret value is converted into multiple blocks each of 8-bits. XOR operation is carried out in between the bits contained in each block of secret value (DV[]) and bits of ciphertext (CT[]) in a cumulative manner and the result is converted into ASCII code from where the decrypted character is generated. Similar activity is carried out for decrypting entire ciphertext.

5.3.3. Implementation with Experiment Results

Encryption is carried out using eighth palindrome number which is generated with a forward movement from user defined base value 10. 31871 milliseconds are needed for encryption or decryption using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM. Table 5.5 shows the encryption and decryption results.

Table 5.5: Content of Plain Text, Ciphertext and Decrypted Text File

Content of Plain Text File (PT.txt)	**Content of Ciphertext File (CT.txt)**	**Content of Decrypted Text File (DT.txt)**
abcdefghijklmnopqrstuvwxyz	WTURSPQ^_\]Z[XYFGDEBC@ANOL	abcdefghijklmnopqrstuvwxyz

Table 5.6 shows the execution result of the algorithm on different types of files (.com, .exe, .txt, .dll, .sys) using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM.

Table 5.6: Execution Results of PAGO Algorithm for Different File Types

Name of the Plain Text File	Size of Plain Text File (Byte)	Size of Encrypted File (Byte)	Encryption Time (Milliseconds)	Decryption Time (Milliseconds)
loadfix.com	1131	1131	32994	32953
ReadMe.txt	286	286	28755	28712
cacls.exe	19968	19968	209737	209695
VIAPCI.SYS	2712	2712	51203	51164
MSMH.DLL	19768	19768	184782	184714
graphics.com	19694	19694	188457	188395
LICENSE.TXT	4829	4829	71099	71041
mqsvc.exe	4608	4608	70793	70746
rootmdm.sys	5888	5888	90852	90813
KBDAL.DLL	6656	6656	84256	84217
diskcomp.com	9216	9216	123750	123711
TechNote.txt	9232	9232	107788	107735
label.exe	9728	9728	92089	92024
sffp_mmc.sys	10240	10240	109843	109794
panmap.dll	10240	10240	135343	135317
kb16.com	14710	14710	155442	155397
ROMAN.TXT	14423	14423	145287	145224
shadow.exe	14848	14848	155814	155782
smclib.sys	14592	14592	152006	151959
tcpmib.dll	14848	14848	157777	157739

The relationship between file size and encryption time is graphically represented in Figure 5.8. Encryption or decryption time does not depend on the type of the file rather it depends on the size of the file as encryption is done on bit level.

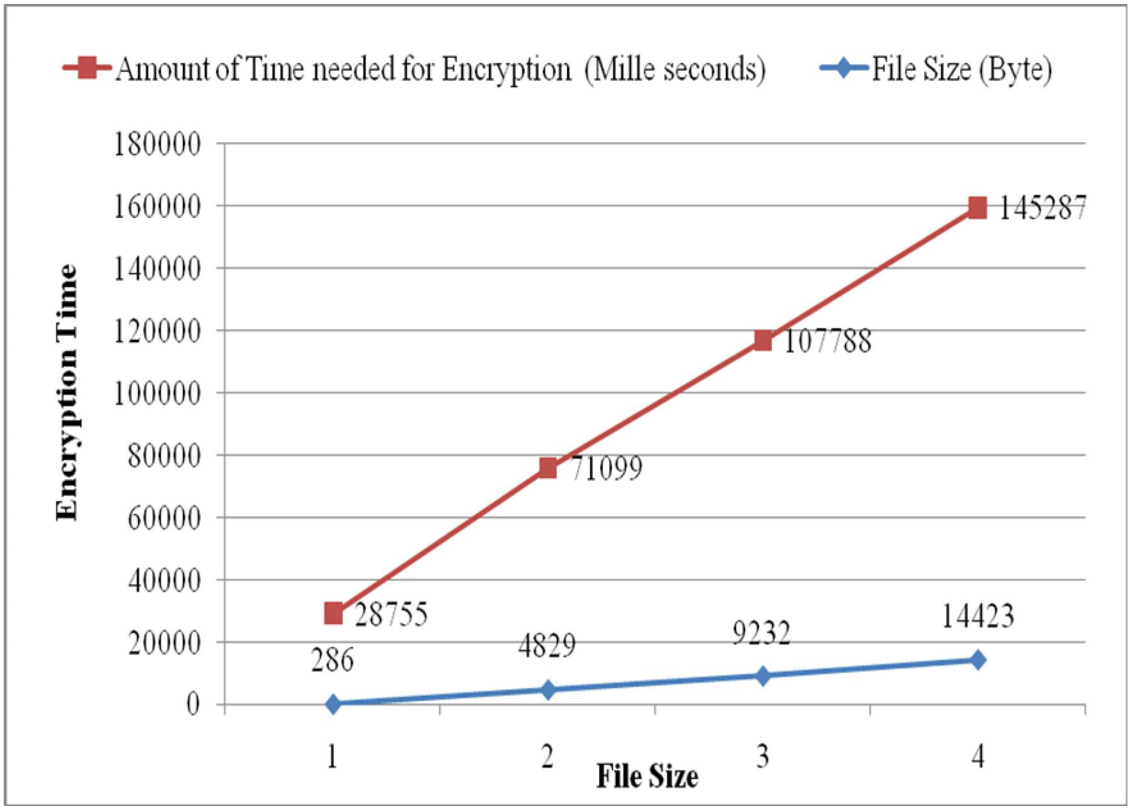

Figure 5.8: Graphical Representation of the Relationship between Encryption Time and File Size

5.3.4. Security analysis for Palindrome number with Alphabetic Group and Operator based text encryption scheme (PAGO)

Table 5.7 and Table 5.8 show the comparison between the results of implemented PAGO scheme with AES-256 algorithms and Serpent scheme respectively. Chi-Square value and degree of freedom value are considered as standard parameters where Chi-Square value and degree of freedom value are calculated as per equation 3.1 and 3.2 respectively mentioned in chapter 3.

Table 5.7: Comparison between Implemented PAGO Algorithm with AES-256 Algorithm in respect of Chi-Square Values and Degree of Freedom Values

File Name	File Size (Byte)	Palindrome number with Alphabetic Group and Operator based text encryption scheme (PAGO)		Results for AES-256 bit Scheme	
		Chi-Square Value	Degree of Freedom	Chi-Square Value	Degree of Freedom
loadfix.com	1131	51633.503906	166	1883.161865	154
graphics.com	19694	787191.562500	254	97162.914063	251
diskcomp.com	9216	168168.656250	246	58842.808594	255
ReadMe.txt	286	1863.688232	57	244.757965	203
LICENSE.TXT	4829	49722.597656	107	6803.458496	255
TechNote.txt	9232	177751.859375	254	12154.286133	251
cacls.exe	19968	5177798.000000	252	133238.406250	251
mqsvc.exe	4608	1348575.625000	215	27239.460938	255
label.exe	9728	1419081.750000	255	70443.242188	252
VIAPCI.SYS	2712	48514.355469	202	6576.918457	255
rootmdm.sys	5888	1103309.625000	254	27792.208984	252
sffp_mmc.sys	10240	1364625.500000	253	60532.246094	251
MSMH.DLL	19768	1278296.375000	255	33590.390625	255
KBDAL.DLL	6656	874256.062500	229	46438.960938	255
panmap.dll	10240	1311916.125000	254	78803.070313	252

Table 5.8: Comparison between Implemented PAGO Algorithm with Serpent Algorithm in respect of Chi-Square Values and Degree of Freedom Values

File Name	File Size (Byte)	Palindrome number with Alphabetic Group and Operator based text encryption scheme (PAGO)		Results for Serpent Scheme	
		Chi-Square Value	Degree of Freedom	Chi-Square Value	Degree of Freedom
loadfix.com	1131	51633.503906	166	2336.749023	250
graphics.com	19694	787191.562500	254	19694.000000	253
diskcomp.com	9216	168168.656250	246	74130.414063	255
ReadMe.txt	286	1863.688232	57	270.753754	172
LICENSE.TXT	4829	49722.597656	107	7000.271484	255
TechNote.txt	9232	177751.859375	254	20356.785156	253
cacls.exe	19968	5177798.000000	252	19968.000000	252
mqsvc.exe	4608	1348575.625000	215	112697.609375	255
label.exe	9728	1419081.750000	255	87114.078125	254
VIAPCI.SYS	2712	48514.355469	202	9424.166016	255
rootmdm.sys	5888	1103309.625000	254	31769.546875	250
sffp_mmc.sys	10240	1364625.500000	253	63553.390625	255
MSMH.DLL	19768	1278296.375000	255	45497.882813	255
KBDAL.DLL	6656	874256.062500	229	62231.351563	255
panmap.dll	10240	1311916.125000	254	102520.625000	253

Figure 5.9 and Figure 5.10 show the graphical comparison between the results of implemented PAGO scheme with AES-256 algorithm and Serpent scheme in respect of Chi-Square value and degree of freedom value respectively.

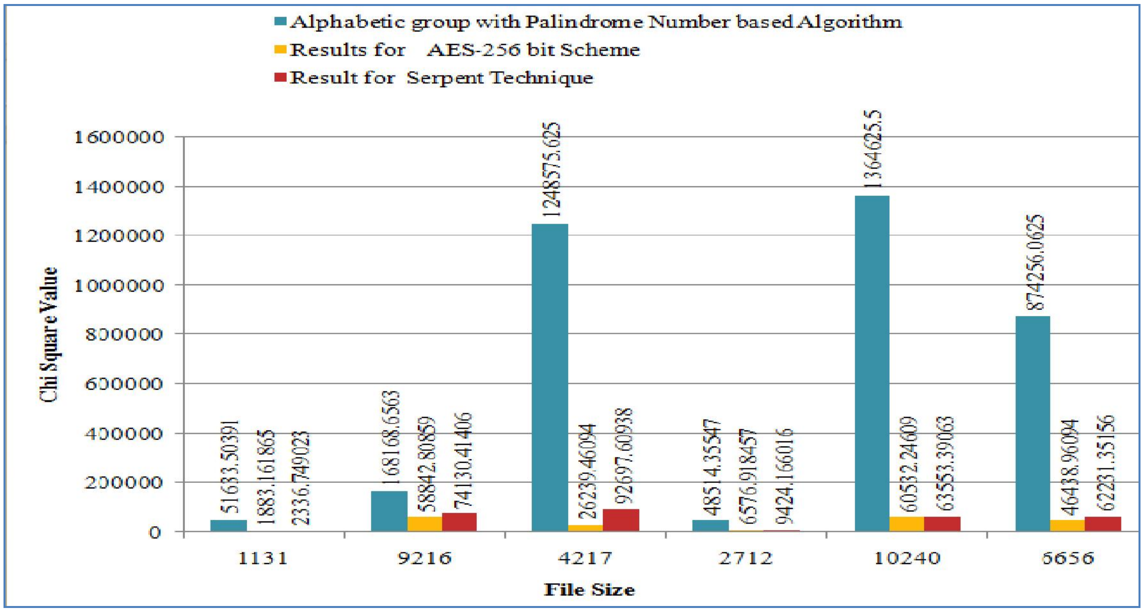

Figure 5.9: Comparison between PAGO Algorithm, AES and Serpent Algorithm in respect of Chi-Square Value

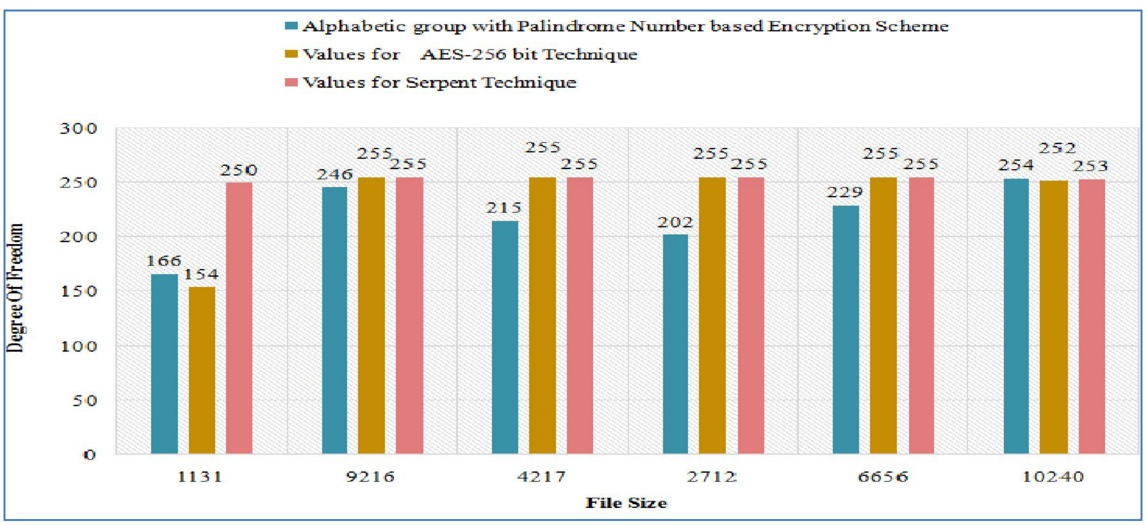

Figure 5.10: Comparison between PAGO Algorithm, AES and Serpent Algorithm in respect of Degree of Freedom

Satisfactory results are shown by PAGO scheme in respect of Chi-Square value and degree o f freedom value. Large key size imposes great security on PAGO scheme.

5.4. Armstrong and Perfect number with Cipher Sequencing based text encryption scheme (APCS)

In the APCS[3] scheme the focus is imposed to implement a secret procedure to derive secret value from the private key and applying it for encryption rather than securing the actual private key value. Secret value is generated by fetching the N^{th} Armstrong or perfect number where N is user define and a positive number. Counting of N^{th} term is started form the user defined base value towards either forward or backward direction. An additional level of security has been imposed on the existing scheme by applying user defined sequence at the time of storing encrypted character blocks in the ciphertext file. Changing of base value, N^{th} term generates different Armstrong or Perfect number which is used as the secret value. A user defined sequence is applied for storing the encrypted characters in ciphertext file corresponding to character sequence in plain text. Thus an attempt is made to enhance the security. Figure 5.11 represents overall procedure for Armstrong and Perfect number with Cipher Sequencing based text encryption scheme (APCS).

Formation of plain text is done by converting each character into 8-bit binary representation. Generate private key by converting the inputs into binary representation and storing the values into the appropriate blocks of key.

↓

Derive the secret value by generating N^{th} Armstrong/ Perfect number with a forward or backward movement from the user defined base value present in private key.

[3] Presented in **IETE Zonal Seminar on ICT in present Wireless Revolution: Challenges and Issues (ICTWR-2013),** pp. 56-65, with title An Approach of Bit-level Private-key Encryption Scheme based on Armstrong Number or Perfect Number with Ordering of Block Ciphers in Selective Mode

Perform XOR operation block wise between secret value and plain text's characters. Thus generate the ciphertext and arrange them as per user mentioned sequence [PRONE, PRENO, CRONE, CRENO] in the final encrypted file.

Ciphertext and private key are shared to the receiver. The reverse process is used for decryption where plain text is generated from ciphertext by performing XOR between the bits of ciphertext and secret value.

Figure 5.11: Overall procedure for Armstrong and Perfect number with Cipher Sequencing based text encryption scheme (APCS)

Section 5.4.1 describes the special number types which have been used for the current algorithm. Section 5.4.2 and section 5.4.3 describe the encryption and decryption process respectively. Section 5.4.4 discusses the experiment results. Security analysis of the algorithm is represented in section 5.4.5.

5.4.1. Special Number Types

A. Armstrong Number

Detailed description is given in section. 1.5.3 of chapter 1 (Different Number Types)

B. Perfect Number

Detailed description is given in section 1.5.5 of chapter 1 (Different Number Types)

5.4.2. Encryption Process

A. Formation of Plain Text

Step 1: Each character from inputted file is converted into 8-bit binary representation and stored into an array called PT[].

B. Generation of Private Key

Step 1: Read and convert the user inputs into a predefined size of numbers of bits as per their corresponding block size of the private key. Store the bit values in an array called key[] with a size of 256.

Size of the private key is 256 bits and five blocks are present in private key. The first block defines the choice between Armstrong number and perfect number. The second block defines the choice between forward and backward movement. The N^{th} term for Armstrong or perfect number is stored in the third block. The fourth block holds the user defined base value. The fifth block stores the value which selects the choice of cipher block sequencing methods by user for repositioning the encrypted character in the ciphertext file. Figure 5.12 represents the block diagram of the private key.

1st block	2^{nd} block	3^{rd} block	4^{th} block	5^{th} block
Selection Code for Armstrong Number or Perfect Number	Selection code for Forward or Backward movement	Value of N^{th} term for Armstrong Number or Perfect Number	User defined Base Value	Choice of cipher block sequencing method for repositioning character blocks in ciphertext
1 bit	1 bit	102 bits	150 bits	2 bits

Figure 5.12: Structure of 256 bits Private Key

C. Generation of Secret Value for Encryption

Step 1: Determine the choice of number (Armstrong or Perfect) and choice of movement (forward or backward) from the values of the first and second block of private key respectively. Generate N^{th} Armstrong or Perfect number by making a backward or forward movement from the base value. Thus the secret value is generated.

Step 2: Convert the secret value into binary representation and store it into an array called DV[].

D. Formation of Ciphertext using XOR Operation

Step 1: Cumulative XOR operation is carried out in between the array PT[] and DV[]. As the size of PT[] is 8-bits and size of DV[] is defined by the secret value, so XOR is performed for multiple times. Array name IR[] with a size of 8 stores intermediate result where the final result is stored in the array EN[] of size 8. ASCII value is generated from array EN[] and from there an encrypted character is constructed. The encrypted character is stored in intermediate cipher text file.

E. Sequencing of Encrypted Character Block

The intermediated encrypted file is split into blocks with a fixed size assigned with a sequence number (1.2…n) (where n is a positive integer). These blocks are repositioned in the final encrypted file as per the cipher block sequencing method is chosen by the user. Final encrypted file (ciphertext) and the private key are shared to the receiver. The descriptions of the cipher block sequencing methods and selection codes are discussed in Table 5.9.

Table 5.9: Description of Cipher Block Sequencing Methods

Selection Code	Name of Cipher Block Sequencing Method	Detail Description of Sequencing Method
00	PRONE (Positional Reverse Odd Normal Even)	Detailed description is given in section 8.3.2 of chapter 8 (Cipher & Pixel Sequencing Methodologies)
10	CRONE (Continuously Reverse Odd Normal Even)	
01	PRENO (Positional Reverse Even Normal Odd)	
11	CRENO (Continuously Reverse Even Normal Odd)	

5.4.3. Decryption Process

A. Repositioning of Encrypted Character's Block in Ciphertext File

Blocks of ciphertext file is repositioned using the same sequencing algorithm used at the time of encryption (using 5.4.2.E). The actual sequence of encrypted characters corresponding to character sequence in the plain text file is obtained in this manner.

B. Conversion of Ciphertext

Step 1: Convert each character of ciphertext file into bits and store the value into an array called CT[] with a size of 8. Carry out the similar activities for all characters of ciphertext file.

C. Generation of Secret Value for Decryption

Secret value is generated from private key using 5.4.2.C and convert that value in binary representation store it into an array called DV[].

D. Generation of Decrypted Text using XOR Operation

Step 1: Cumulative XOR operation is carried out between the array CT[] and DV[]. Array IR[] and array DT[] of size 8 store the intermediate result and final result respectively. ASCII code is generated from array DT[] and from there the decrypted character is also generated which is stored into the decrypted text file.

5.4.4. Experiment Results and Discussion

Encryption is done using the secret value which is derived by fetching the 8^{th} Armstrong number with the forward movement from user defined base value 123. PRONE ordering sequence is applied for character repositioning. 18922 milliseconds are needed for encryption using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM. Table 5.10 shows encryption results.

Table 5.10: Content of Plain Text, Intermediate Encrypted, Ciphertext and Decrypted Text File

Content of Plain Text File (PT.txt)	Content of Intermediate Encrypted File (EN_I.txt)	Content of Ciphertext File(CT.txt)	Content of Decrypted Text File(DT.txt)
qwertyuioplk	x~l{}p|`fyeb	{l~x}p|`beyf	qwertyuioplk

Table 5.11 represents the execution result of the APCS algorithm on different types of files (.com, .exe, .txt, .dll, .sys) using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM.

Table 5.11: Execution Results of APCS Algorithm on Different File Types

File Name	Size of Plain Text File (Byte)	Encrypted File Size (Byte)	Encryption Time (Milliseconds)	Decryption Time (Milliseconds)
loadfix.com	1131	1131	42187	42112
ReadMe.txt	286	286	64000	63973
WINSTUB.EXE	578	578	25172	25119
VIAPCI.SYS	2712	2712	92843	92811
iconlib.dll	2560	2560	58359	58325
README.COM	4217	4217	55000	54971
LICENSE.TXT	4829	4829	69515	69482
mqsvc.exe	4608	4608	55250	55213
rootmdm.sys	5888	5888	115625	115586
KBDAL.DLL	6656	6656	122609	122559
diskcomp.com	9216	9216	202640	202617
TechNote.txt	9232	9232	212578	212527
label.exe	9728	9728	212000	211971
sffp_mmc.sys	10240	10240	236219	236173
panmap.dll	10240	10240	308563	308512
kb16.com	14710	14710	408031	408013
ROMAN.TXT	14423	14423	367453	367417
shadow.exe	4848	4848	337781	337754
smclib.sys	14592	14592	374937	374915
tcpmib.dll	14848	14848	388922	388910

The relationship between encryption time and file size is graphically represented in Figure 5.13. Encryption or decryption time is independent of file type rather it depends on file size.

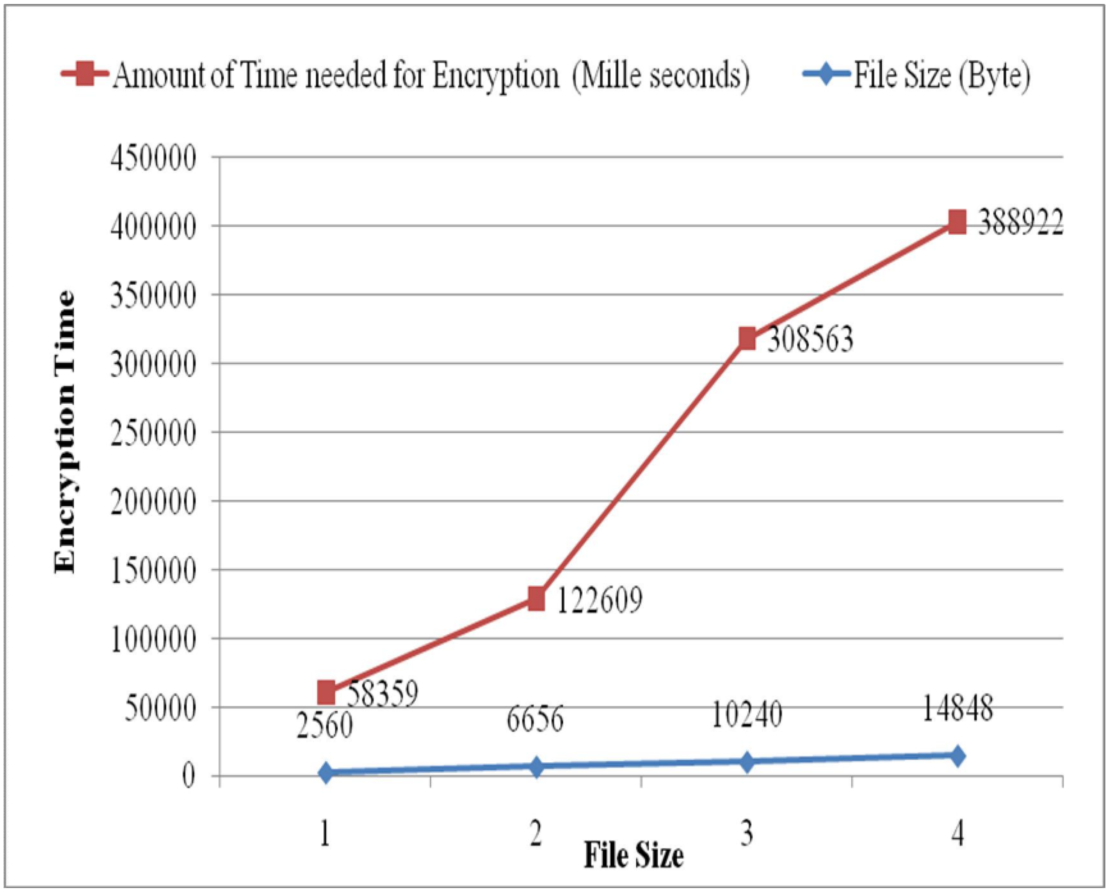

Figure 5.13: Representation of Relationship between File Size and Encryption Time

5.4.5. Security analysis of Armstrong and Perfect number with Cipher Sequencing based text encryption scheme (APCS)

Table 5.12 and Table 5.13 shows the comparison between the results of the APCS scheme with AES-256 and Blowfish scheme respectively. A comparison is done in respect of Chi-Square and degree of freedom 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 5.12: Comparison between APCS Algorithm with AES-256 Algorithm in respect of Chi-Square Values and Degree of Freedom Values

File Name	**File Size (Byte)**	**Armstrong and Perfect number with Cipher Sequencing based text encryption scheme (APCS)**		**Result for AES-256 Scheme**	
		Chi-Square Value	**Degree of Freedom**	**Chi-Square Value**	**Degree of Freedom**
iconlib.dll	2560	4251102	162	5869402.0000	255
KBDAL.DLL	6656	6457562.5000	249	1422929.0000	245
panmap.dll	10240	4774061.5	253	3118043.0000	251
VIAPCI.SYS	2712	1986680.7500	254	1259225.0000	251
rootmdm.sys	5888	4113326.5000	250	5683317.0000	255
sffp_mmc.sys	10240	3366700.2500	254	3054482.0000	251
WINSTUB.EXE	578	580.833313	73	5377464.0000	255
mqsvc.exe	4608	5081017.0000	251	16715717.000	255
label.exe	9728	571898.21875	253	433017.0000	250
ReadMe.txt	286	286	66	123631272.000	255
LICENSE.TXT	4829	4829	145	28335868.0000	255
TechNote.txt	9232	9232	156	19943582.0000	255
loadfix.com	1131	527132.03125	252	496330.0000	250
README.COM	4217	493531.46875	250	123514.0000	247
diskcomp.com	9216	283393.25000	251	4464414.0000	255

Table 5.13: Comparison between APCS Algorithm and Blowfish Scheme in respect of Chi-Square Values and Degree of Freedom Values

File Name	File Size (Byte)	Armstrong and Perfect number with Cipher Sequencing based text encryption scheme (APCS)		Result for Blowfish Scheme	
		Chi-Square Value	Degree of Freedom	Chi-Square Value	Degree of Freedom
iconlib.dll	2560	4251102	162	40425.773438	254
KBDAL.DLL	6656	6457562.50000	249	58675.128906	248
panmap.dll	10240	4774061.5	253	78465.289063	251
VIAPCI.SYS	2712	1986680.75000	254	15741.601563	252
rootmdm.sys	5888	4113326.50000	250	33945.304688	249
sffp_mmc.sys	10240	3366700.25000	254	95056.382813	253
WINSTUB.EXE	578	580.833313	73	12396.750000	125
mqsvc.exe	4608	5081017.00000	251	126816.570313	250
label.exe	9728	571898.21875	253	181168.234375	251
ReadMe.txt	286	286	66	241.758804	175
LICENSE.TXT	4829	4829	145	7152.312500	255
TechNote.txt	9232	9232	156	13975.289063	255
loadfix.com	1131	527132.03125	252	5692.211426	242
README.COM	4217	493531.46875	250	11718.753906	249
diskcomp.com	9216	283393.25000	251	61566.996094	255

Figure 5.14 and Figure 5.15 show the graphical comparison between APCS scheme with AES-256 and Blowfish scheme in respect of Chi-Square value and degree of freedom value respectively.

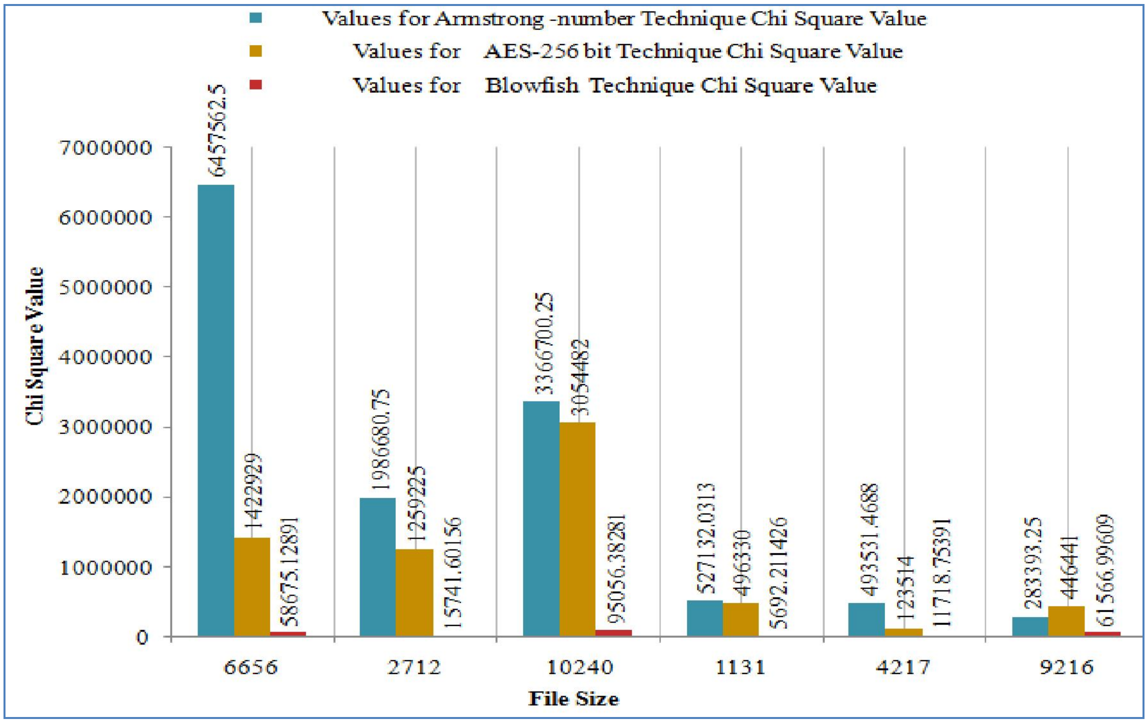

Figure 5.14: Comparison between APCS Algorithm, AES and Blowfish Algorithm in respect of Chi-Square value

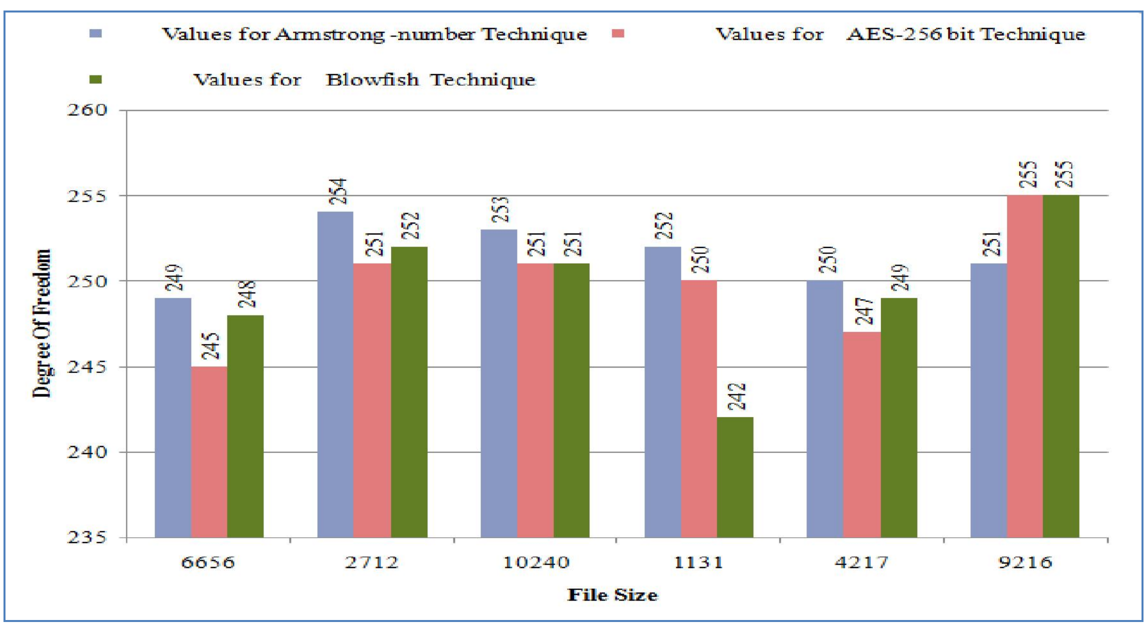

Figure 5.15: Comparison between APCS Algorithm, AES and Blowfish Algorithm in respect of Degree of Freedom value

APCS scheme provides satisfactory results as per the degree of freedom value and Chi-Square value. Security is increased in great extent as the large key size is applied by APCS scheme.

5.5. Amicable number with Cipher Sequencing based text encryption scheme (ACS)

Distribution of key through public channels without interpretation is very hard to implement, so in ACS[4] scheme, the focus is imposed to build secret procedures which generate secret value from the private key for encryption rather than using the private key value directly. Generation of secret value is carried out by fetching the 1st or 2nd item from N^{th} amicable number pair where N is user defined positive number. Counting of N^{th} amicable number is started from user defined base value towards its forward or backward direction where the direction of movement is defined by user input. An additional layer of security is implemented by applying user defined sequence for storing encrypted character block in ciphertext file. So whenever the base value, N^{th} term is changed corresponding Armstrong number is also changed. So a new secret value is generated. Prediction of encrypted character sequence corresponding to source character sequence is difficult as user defined cipher block sequencing is applied for storing encrypted characters. Thus an attempt is made to enhance the security. Figure 5.16 represents the overall procedure for Amicable number with Cipher Sequencing based text encryption scheme (ACS).

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.

[4] Published in **International Journal of Computer sciences and Engineering (ICSE), UGC approved journal,** Volume 6, Issue 5, pp. 34 – 41, DOI: https://doi.org/10.26438/ijcse/v6i5.3441, with title A Private Key Encryption Scheme based on Amicable Number with User defined Cipher Block Sequencing Techniques

Derived the secret value by making user defined selection of number and movement from base value directed by primary key and perform bitwise XOR between plain text character and the secret value. Thus intermediate encrypted file is generated.

↓

Rearrange all encrypted characters as per one of cipher block sequencing technique (EEFR, EEFL, EOFR, EOFL) chosen by the user and it generates the final encrypted file.

↓

The reverse process is carried out for decryption. All the characters of the ciphertext file are repositioned using a sequencing algorithm which is applied at encryption time. Plain text is generated from ciphertext by performing XOR between the bits of ciphertext and secret value.

Figure 5.16: Overall Procedure for Amicable number with Cipher Sequencing based text encryption scheme (ACS)

Section 5.5.1 represents the description of different terminology. Section 5.5.2 and section 5.5.3 discusses encryption and decryption process respectively. Experiment results and security analysis of amicable number based encryption scheme are represented in section 5.5.4 and section 5.5.5 respectively.

5.5.1. Different Terminology

A. Amicable Number

Detailed description is given in section. 1.5.4 of chapter 1 (Different Number Types)

B. Cipher Block Sequencing Techniques

Different cipher block sequencing methods are discussed in Table 5.14.

Table 5.14: Description of Different Cipher Block Sequencing Techniques

Selection Code	Name of Cipher Block Sequencing Technique	Description of Sequencing Techniques
00	EEFR(Exchange Even From Right)	Detail description is given in section 8.3.2 of chapter 8 (Cipher & Pixel Block Sequencing Methodologies)
10	EOFR(Exchange Odd From Right)	
01	EEFL(Exchange Even From Left)	
11	EOFL(Exchange Odd From Left)	

5.5.2. Encryption Process

A. Generation of Plain Text

Read and convert each single character into 8-bit binary representation from plain text file till all the characters are visited and store the value into an array called PT[].

B. Formation of Private Key

Step 1: Read and convert the user inputs into bits corresponding to their respective block size in private key. Store the bit values into an array called KEY[] of size 256.

Private key has five numbers of blocks and the total length of the key is 256 bits. The first block defines the selection of forward or backward movement from the base value. The second block holds Nth numbered pair of the amicable number. The third block holds the value which makes the selection of either first or second number from N^{th} amicable number pair. The fourth block holds base value. Value of the fifth block defines the selection of cipher block sequencing techniques applied for repositioning the characters of intermediate encrypted file Figure 5.17 represents the structure of private key.

1st block	2nd block	3rd block	4th block	5th block
Selection code for forward or backward movement	N^{th} term for amicable number pair	Selection code for first or second number from amicable number pair	Base value	Selection code for cipher block sequencing scheme for repositioning the characters of intermediate encrypted file
1 bit	102 bits	1 bit	150 bits	2 bits

Figure 5.17: Structure of 256 bits Primary Key

C. Generation of Secret Value

Step 1: Determine the N^{th} pair of an amicable number and the movement (forward and backward) from the values present in the first and second block of the private key respectively. Fetch the N^{th} amicable pair with a forward or backward movement from the base value.

Step 2: Selection of first or second amicable number from N^{th} amicable pair is determined as per the value present in the third block of the primary key. Selected number is considered as secret value and an array called DV[] keeps the binary representation of that value.

D. Generation of Ciphertext using XOR operation

XOR operation is performed in between the array DV[] and PT[] in a cumulative manner. As the size of DV[] depends on secret value and size of PT[] is 8-bits, so bitwise XOR operation is carried out for multiple times. The intermediate result is stored in array IR[] with a size of 8 and array EN[] stores the final result. Value of array EN[] generates the ASCII code from there an encrypted character is constructed. All encrypted characters are kept in intermediate ciphertext file.

E. Repositioning of Intermediate Encrypted File and Generation of Ciphertext

An intermediated encrypted file is divided into multiple blocks with a fixed size. Those blocks are assigned with sequence numbers (1.2….K where K is a positive integer)). The blocks of characters are repositioned as per user defined cipher block sequencing schemes selected based on the value present in the fifth block of the primary key. Thus generates a final encrypted file (ciphertext) which is shared to the receiver along with the private key.

5.5.3. Decryption Process

A. Re-sequencing of Character's Block from Ciphertext File

Encrypted character blocks from ciphertext file are repositioned using the same cipher block sequencing scheme applied at encryption time (using 5.5.2.E). Encrypted character's actual sequence corresponding to plain text's character sequence is achieved by this procedure.

B. Conversion of Ciphertext

Step 1: Read and convert each character into 8-bit binary representation from the ciphertext file until all the characters are visited. Store the bit values of the character into an array called CT[].

C. Formation of Secret Value for Decryption

Generation of secret value is carried out from private key using 5.5.2.C. Secret value is converted into binary representation and store the value into an array called DV[].

D. Formation of Decrypted Text using XOR Operation

Step 1: Cumulative bitwise XOR operation is performed between array CT[] and array DV[]. Intermediate and final results are stored in array IR[] and array DT[] respectively. From array DT[], the decrypted character is derived. Decrypted characters are stored in the decrypted text file.

5.5.4. Experiment Result and Discussions

Generation of secret value is done by fetching the first item from 8^{th} pair of the amicable number set where forward movement is made from the base value 200. Secret value (amicable number) is 17296 and EOFL cipher sequencing is applied as selection code is 11. 131530 milliseconds are needed for encryption using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM. Table 5.15 represents the result of encryption.

Table 5.15: Content of Plain Text, Ciphertext, Intermediate Encrypted and Decrypted Text File

Content of Plain Text File	Content of Intermediate Encrypted Text File	Content of Ciphertext File	Content of Decrypted Text File
AVKZ=-(@xpqt1058	¨¿¢³ÔÄÁ©????ØÙÜÑ	????ÔÄÁ©¨¿¢³ØÙÜÑ	AVKZ=-(@xpqt1058

Table 5.16 represents the execution result of amicable number based encryption scheme for different file types (.com, .exe, .txt, .dll, .sys) using a computer with Core 2 Duo 2.20 GHz processor and 1.00 GB RAM.

Table 5.16: Execution Results of ACS Scheme on Different File Types

File Name	File Size (KB)	Size of Encrypted File (KB)	Time needed for Encryption (Milliseconds)	Time needed for Decryption (Milliseconds)
Diskcomp.com	9	9	891368	891324
Iconlib.dll	2.50	2.50	249273	249229
Cacls.exe	19.5	19.5	1907236	1907199
Partmgr.sys	19.2	19.2	1770073	1770025
AAB.txt	19.5	19.5	1744230	1744197
graphics.com	19.2	19.2	2112340	2112313
KBDAL.dll	6.50	6.50	608651	608619
label.exe	9.50	9.50	862988	862947
rootmdm.sys	5.75	5.75	527221	527185
LICENSE.txt	4.71	4.71	432543	432517
kb16.com	14.3	14.3	1383434	1383393
MSMH.dll	19.3	19.3	1923830	1923793
mqsvc.exe	4.50	4.50	406848	406812
sffp_mmc.sys	10.0	10.0	908839	908817
ReadMe.txt	1	1	42826	42783
loadfix.com	1.10	1.10	133367	133314
panmap.dll	10.0	10.0	954704	954658
Shadow.exe	14.5	14.5	1315419	1315411
Smclib.sys	14.2	14.2	1308089	1308051
ROMAN.txt	14.0	14.0	1376958	1376917
README.com	4.11	4.11	402355	402313
tcpmib.dll	14.5	14.5	1423771	1423745
WINSTUB.exe	1	1	112897	112863
VIAPCI.sys	2.64	2.64	244524	244513
TechNote.txt	9.01	9.01	855701	855654

Figure 5.18 represents the relationship between file size and encryption time graphically from where it is mentioned that there is a proportional relationship between file size and encryption time.

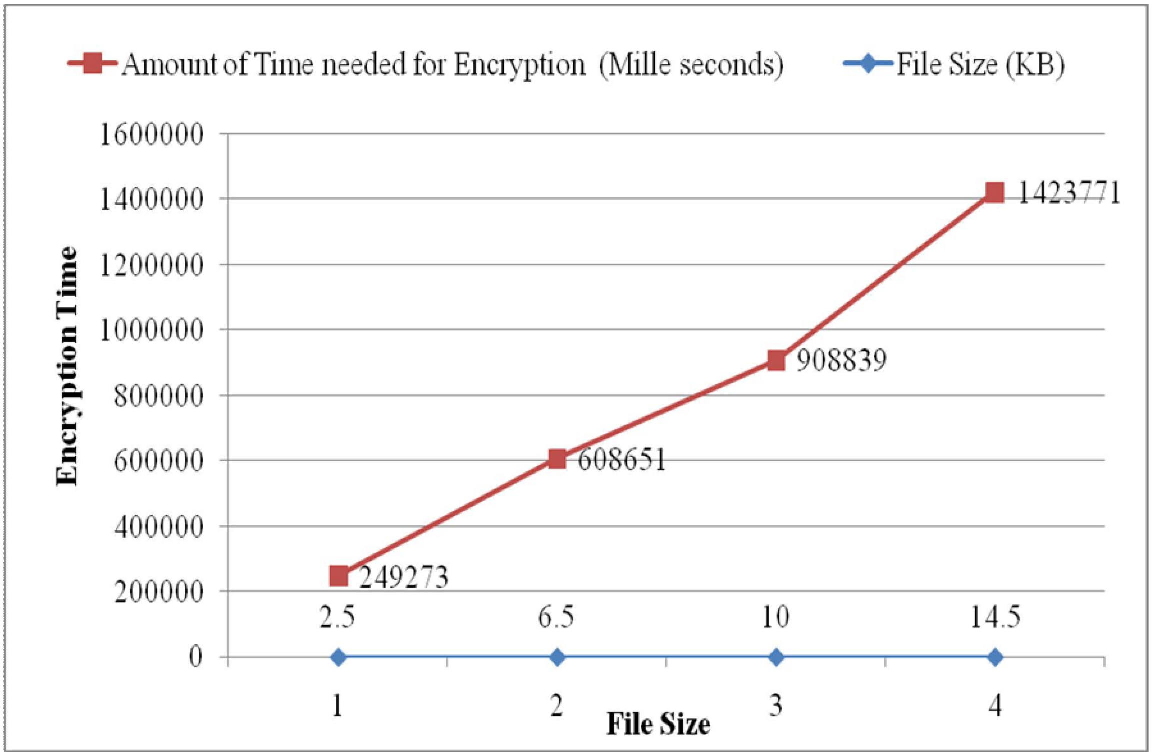

Figure 5.18: Relationship between Encryption Times with File Size of Plain Text

5.5.5. Security Analysis of Amicable number with Cipher Sequencing based text encryption scheme (ACS)

Table 5.17 and Table 5.18 represent comparisons between the ACS scheme with AES-256 and Triple DES scheme respectively. Chi-Square test value and degree of freedom are considered as parameters for comparison where Chi-Square value and degree of freedom value are calculated as per equation 3.1 and 3.2 respectively mentioned in chapter 3.

Table 5.17: Comparison between ACS Algorithm with AES-256 Algorithm in respect of Degree of Freedom Values and Chi-Square Values

File Name	Source File Size (KB)	File Type	Result for Amicable number with Cipher Sequencing based text encryption scheme (ACS)		Result forAES-256 bit Technique	
			Chi-Square Value	Degree of Freedom	Chi-Square Value	Degree of Freedom
diskcomp	9	.com	59162.722656	255	9216.000000	245
graphics	19.2	.com	90349.453125	255	19694.000000	254
kb16	14.3	.com	49740.839844	255	14710.000000	255
loadfix	1.10	.com	1849.756104	254	1131.000000	145
README	4.11	.com	10640.603516	255	4217.000000	242
iconlib	2.50	.dll	6672.274902	255	2560.000000	113
KBDAL	6.50	.dll	44474.777344	255	6656.000000	227
MSMH	19.3	.dll	34499.722656	255	19768.000000	255
panmap	10.0	.dll	75107.367188	255	10240.000000	243
tcpmib	14.5	.dll	96802.875000	255	14848.000000	254
cacls	19.5	.exe	127799.695313	255	19968.000000	252
label	9.50	.exe	76094.953125	255	9728.000000	247
mqsvc	4.50	.exe	29264.796875	255	4608.000000	208
Shadow	14.5	.exe	66915.531250	255	14848.000000	255
WINSTUB	1	.exe	634.412964	232	578.000000	45
partmgr	19.2	.sys	112557.718750	255	19712.000000	255
rootmdm	5.75	.sys	26591.115234	255	5888.000000	231
sffp_mmc	10.0	.sys	58059.800781	255	10240.000000	251
Smclib	14.2	.sys	58216.992188	255	14592.000000	254
VIAPCI	2.64	.sys	7212.227051	255	2712.000000	190
AAB	19.5	.txt	7609.227051	255	20000.000000	255
LICENSE	4.71	.txt	6572.482910	255	4829.000000	73
ReadMe	1	.txt	242.766479	201	296.000000	33
ROMAN	14.0	.txt	22061.707031	255	14423.000000	82
TechNote	9.01	.txt	13500.411133	255	9232.000000	78

Table 5.18: Comparison between ACS Algorithm with Triple DES Scheme in respect of Chi-Square Values and Degree of Freedom Values

File Name	File Size (KB)	Result for Amicable number with Cipher Sequencing based text encryption scheme (ACS)		Result for Triple DES Scheme	
		Chi-square Test Value	Degree of Freedom	Chi-square Test Value	Degree of Freedom
diskcomp	9	59162.722656	255	60416.285156	255
graphics	19.2	90349.453125	255	90116.484375	255
kb16	14.3	49740.839844	255	49325.226563	255
KBDAL	6.50	44474.777344	255	85571.742188	255
MSMH	19.3	35699.722656	255	34495.363281	255
panmap	10.0	75107.367188	255	182137.109375	255
label	9.50	76094.953125	255	161537.390625	255
mqsvc	4.50	29264.796875	255	20580.156250	255
WINSTUB	1	634.412964	127	564.752014	127
rootmdm	5.75	36591.115234	255	33774.203125	255
sffp_mmc	10.0	58059.800781	255	132770.812500	255
VIAPCI	2.64	7212.227051	255	14035.050781	255
AAB	19.5	7609.227051	255	7314.505371	255
LICENSE	4.71	6572.482910	255	6515.068359	255
ROMAN	14.0	22061.707031	255	34809.195313	255

Figure 5.19 and Figure 5.20 shows the comparison between ACS scheme with AES-256 and Triple DES scheme graphically in respect of Chi-Square and degree of freedom value respectively.

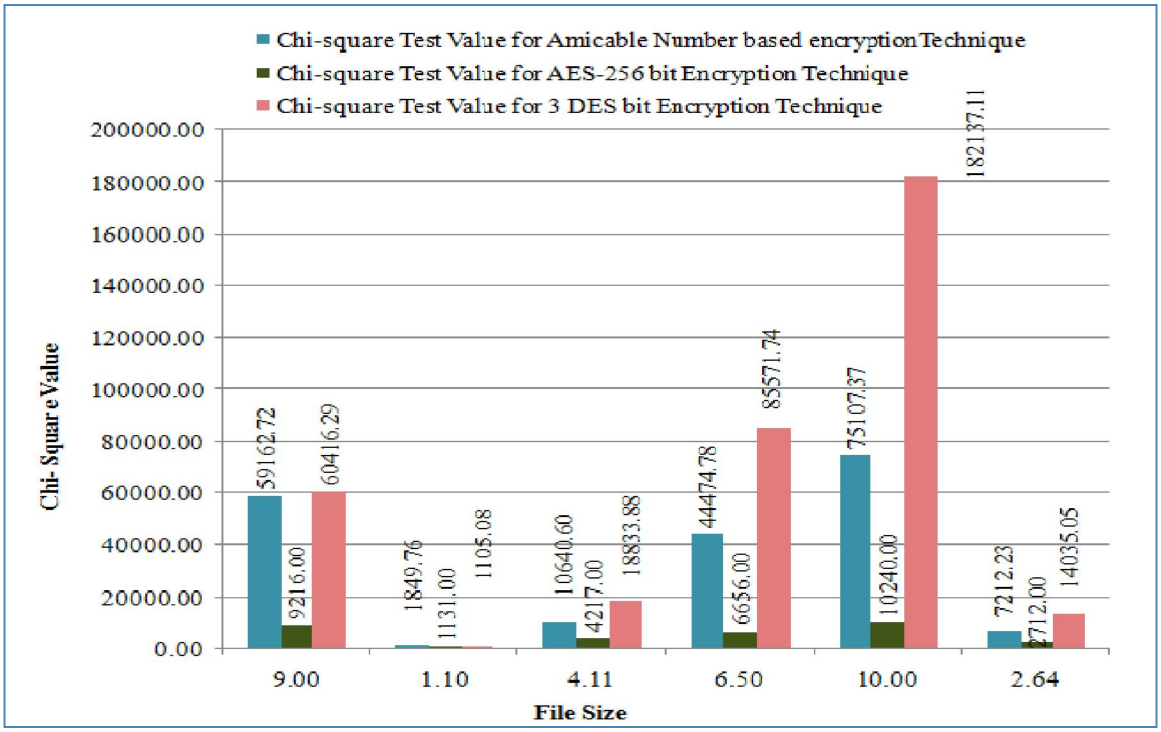

Figure 5.19: Comparison between ACS Algorithm, AES and Triple DES Algorithm in respect of Chi-Square value

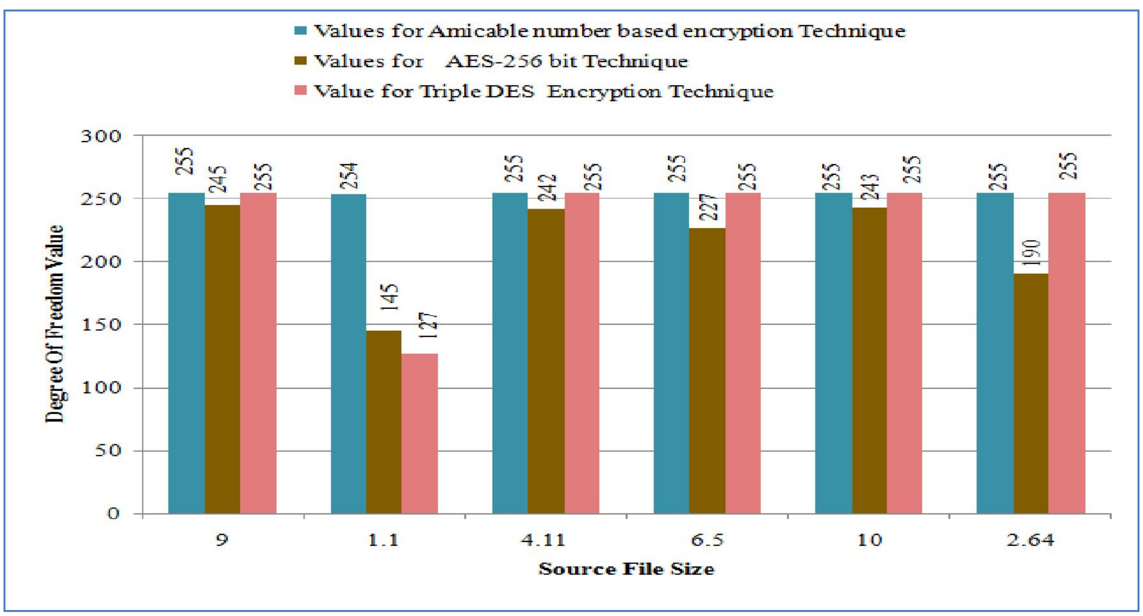

Figure 5.20: Comparison between ACS Algorithm, AES and Triple DES Algorithm in respect of Degree of Freedom value

Satisfactory performances are observed for the ACS scheme as per the degree of freedom value and Chi-Square value. Large key size enhanced the security in great extent for ACS scheme.

5.6. Conclusion

The secret value is counted from user defined base value for the implemented text based encryption schemes. So changing of base value, the N^{th} term, alphabetic group and combination sequence generate different secret value. Thus security is increased.

The secret value generation procedure is secured and private between sender and receiver. So if the private key is compromised still the system is secured. Thus the security is increased to a great extent.

As the placement ordering sequence of encrypted characters is specified by user choice, so prediction of an encrypted character corresponding to its source character is very difficult. Thus an attempt is made to enhance the security.

As the encryption is carried out in bit level so the encryption time does not depend on file type and the encrypted file size is same with plain text. So no additional space is needed for the implemented schemes.

As compared with AES, Twofish, Triple DES, Blowfish and Serpent schemes, implemented schemes provide a great result in respect of Chi-Square value and degree of freedom value. So it may be concluded that the newly implemented text encryption schemes may provide very satisfactory level of security for text encryption.