MULTIPLE ENCRYPTION

using Caesar Cipher, XOR Cipher, AES, Blowfish Cipher

A REPORT

Submitted by

PRAGNA PULIPATI (17MIS1044)
K SRI HARIKA (17MIS1149)

in the partial fulfilment for the award

of

M. Tech Software Engineering (Integrated)

School of Computer Science and Engineering

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

JUNE 2020 – AUGUST 2020
(FALL SEMESTER 2020-2021)

Course Title: Information and System Security
Course Code: SWE3002
Submitted to: Prof. Punitha K

OBJECTIVE:

The main objective of this project is to gain knowledge about various methods of encryption and execute our idea of achieving Multiple Encryption. This Multiple Encryption is built using Caesar Cipher, XOR Cipher, Advanced Encryption Standard (AES) Cipher, Blowfish Cipher.

- To study the importance of Encryption
- To understand various Encryption Methodologies and algorithms
- To implement Multiple Encryption using various Cipher algorithms.

INTRODUCTION:

Encryption is a process that encodes a message or file so that it can be only be read by certain people. Encryption uses an algorithm to scramble, or encrypt, data and then uses a key for the receiving party to unscramble, or decrypt, the information. The message contained in an encrypted message is referred to as plaintext. In its encrypted, unreadable form it is referred to as ciphertext. Encryption uses algorithms to scramble your information. It is then transmitted to the receiving party, who is able to decode the message with a key. There are many types of algorithms, which all involve different ways of scrambling and then decrypting information.

Multiple encryption is the process of encrypting an already encrypted message one or more times, either using the same or a different algorithm. The algorithms used here are:

- Caesar Cipher, also known as Caesar's cipher, the shift cipher, Caesar's **c**ode or Caesar shift, is one of the simplest and most widely known encryption techniques.
- XOR Cipher is a type of additive cipher, an encryption algorithm that operates according to the XOR principles.
- AES Cipher is based on a design principle known as a substitution–permutation network, and is efficient in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network.
- Blowfish is a symmetric-key block cipher. Blowfish provides a good encryption rate in software and no effective cryptanalysis of it has been found to date.

KEYWORDS:

Multiple Encryption, Multi-level Encryption, Caesar Cipher, XOR Cipher, AES Cipher, Blowfish Cipher, Key generation, Plain Text, Cipher Text, Encrypted Text, Encryption, Algorithm, Feistel structure.

FEASIBILITY STUDY:

Advantages:

- Increased security
- Automatic key generation
- Hazzle free text encryption

Disadvantages:

- The size constraint stands as a disadvantage for this proposal.
- Large inputs cannot be encrypted with the help of this.

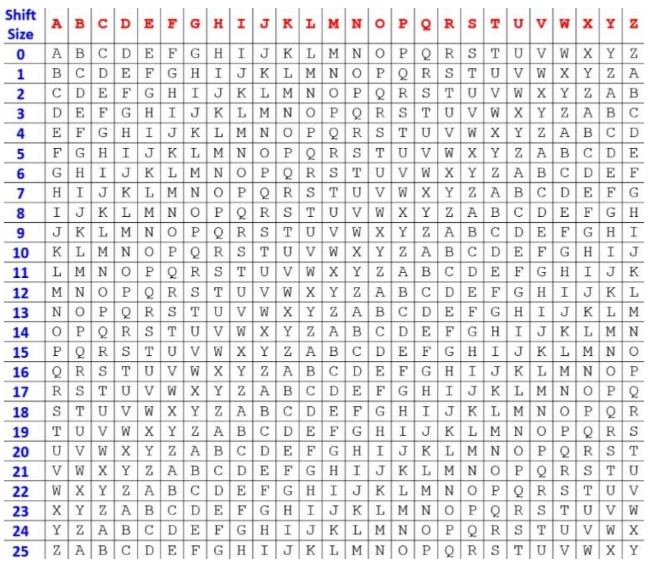
REQUIREMENT ANALYSIS:

- JDK – 13.0.2
- 64-bit Windows Operating System

MODULES:

1. Caesar Cipher
2. XOR Cipher
3. AES Encryption
4. Blowfish Cipher

CAESAR CIPHER: It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.



Shift Size	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 1: Caesar Cipher Shift Table

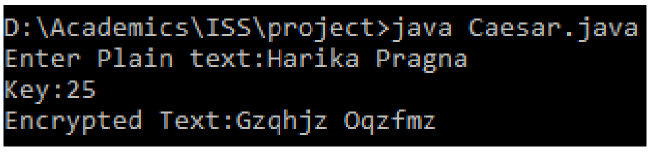
Code:

```
import java.util.*;
class Caesar{
public static String CaesarCipher(String input, int k1){
String result= new String();
char ch1[]=input.toCharArray();
for(int i=0; i<input.length(); i++){
if(Character.isLetter(ch1[i])){
if (Character.isUpperCase(input.charAt(i))){
char ch = (char)(((int)input.charAt(i) + k1 - 65) % 26 + 65);
result=result+ch;
}
else{
char ch = (char)(((int)input.charAt(i) + k1 - 97) % 26 + 97);
result=result+ch;
}}
else if(ch1[i]==' '){
char ch = ch1[i];
result=result+ch;
}}
return result;
}
public static void main(String args[]) throws Exception{
String input;
int k1;
int max=26;
String l1;
try (Scanner sc= new Scanner(System.in)){
System.out.print("Enter Plain text:");
input = sc.nextLine( );
Random rand = new Random();
k1 = rand.nextInt(max);
System.out.println("Key:"+k1);
l1 = CaesarCipher(input,k1);
```

System.out.println("Encrypted Text:"+l1);

}}}

Output:



```
D:\Academics\ISS\project>java Caesar.java
Enter Plain text:Harika Pragna
Key:25
Encrypted Text:Gzqhjz Oqzfmz
```

XOR : XOR cipher is a type of additive cipher, an encryption algorithm that operates according to the principles:

$A \oplus 0 = A,$

$A \oplus A = 0,$

$(A \oplus B) \oplus C = A \oplus (B \oplus C),$

$(B \oplus A) \oplus A = B \oplus 0 = B,$

where $\oplus$ denotes the exclusive disjunction XOR operation. This operation is sometimes called modulus 2 addition (or subtraction, which is identical). With this logic, a string of text can be encrypted by applying the bitwise XOR operator to every character using a given key. To decrypt the output, merely reapplying the XOR function with the key will remove the cipher.

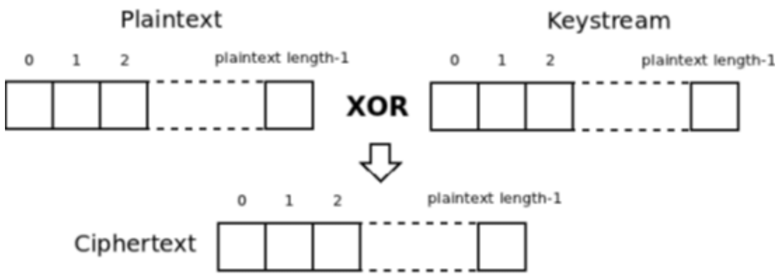
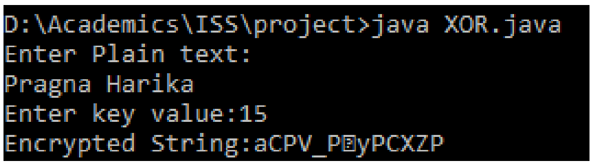


Figure 2: Sketch of XOR Encoding

Code:

```
import java.util.*;
public class XOR {
static String encryptDecrypt(char xorKey,String inputString){
String outputString = "";
int len = inputString.length();
for (int i = 0; i < len; i++){
outputString = outputString + Character.toString((char) (inputString.charAt(i) ^ xorKey));
}
return outputString;
}
public static void main(String[] args){
String sampleString;
try (Scanner sc= new Scanner(System.in)){
System.out.println("Enter Plain text:");
sampleString = sc.nextLine( );
System.out.print("Enter key value:");
char xorKey = sc.next().charAt(0);
System.out.println("Encrypted String:"+encryptDecrypt(xorKey,sampleString));
}}}
```

Output:



```
D:\Academics\ISS\project>java XOR.java
Enter Plain text:
Pragna Harika
Enter key value:15
Encrypted String:aCPV_P⍰yPCXZP
```

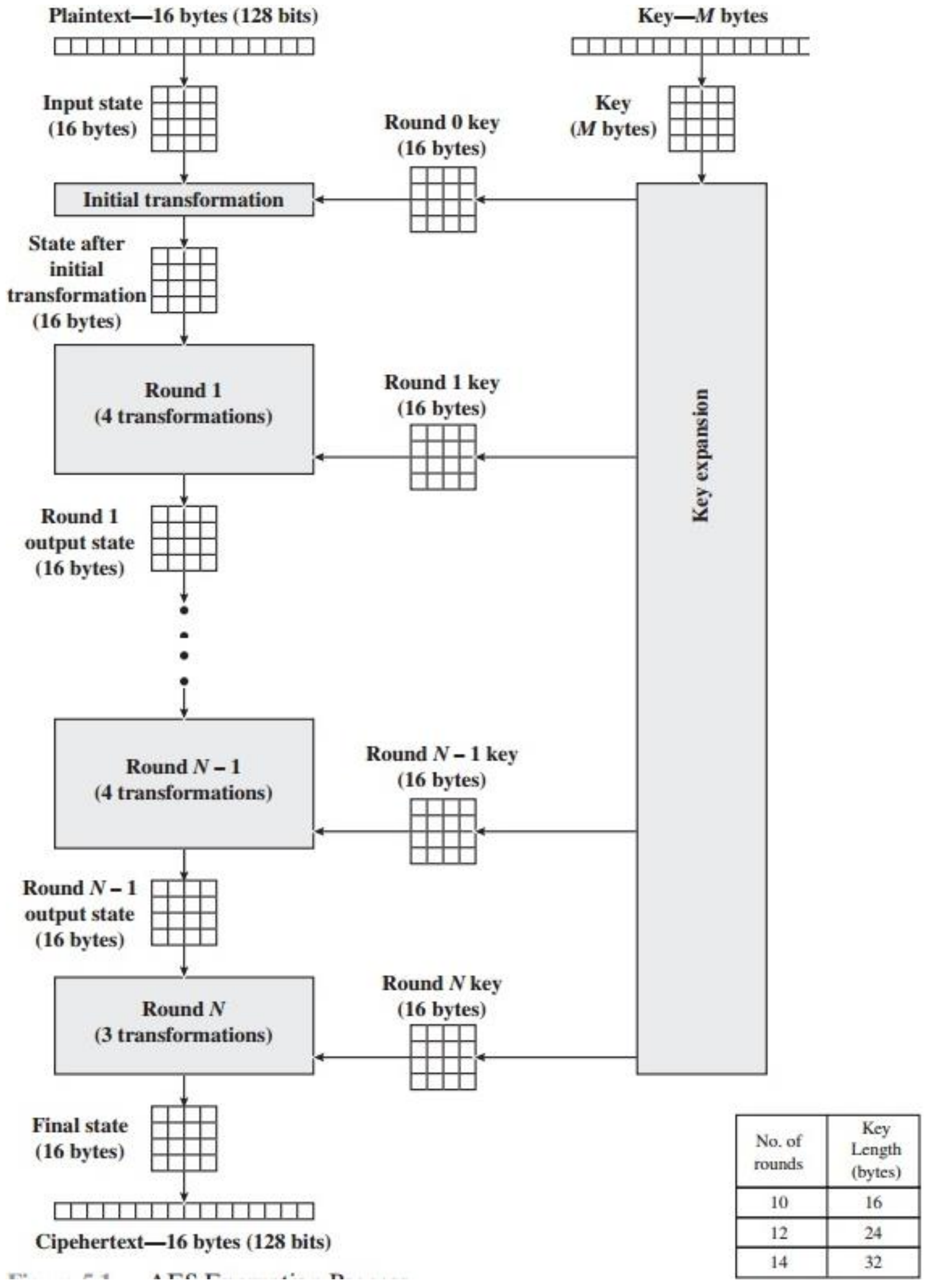
AES: AES operates on a 4 × 4 column-major order array of bytes, termed the state. Most AES calculations are done in a particular finite field. For instance, 16 bytes, are represented as this two-dimensional array.

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

The key size used for an AES cipher specifies the number of transformation rounds that convert the input, called the plaintext, into the final output, called the ciphertext. The number of rounds are as follows:

- 10 rounds for 128-bit keys
- 12 rounds for 192-bit keys
- 14 rounds for 256-bit keys.

Each round consists of several processing steps, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.


Plaintext—16 bytes (128 bits)
Key—M bytes
Input state (16 bytes)
Key (M bytes)
Round 0 key (16 bytes)
Initial transformation
State after initial transformation (16 bytes)
Round 1 (4 transformations)
Round 1 key (16 bytes)
Key expansion
Round 1 output state (16 bytes)
Round N – 1 (4 transformations)
Round N – 1 key (16 bytes)
Round N – 1 output state (16 bytes)
Round N (3 transformations)
Round N key (16 bytes)
Final state (16 bytes)
Cipehertext—16 bytes (128 bits)

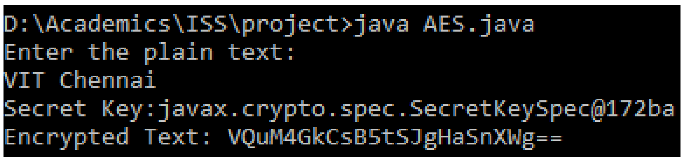
No. of rounds	Key Length (bytes)
10	16
12	24
14	32

Figure 3: AES Algorithm Structure

Code:

```
import java.util.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
public class AES{
static Cipher cipher;
public static void main(String[] args) throws Exception {
try (Scanner sc = new Scanner(System.in)) {
KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
keyGenerator.init(128);
SecretKey secretKey = keyGenerator.generateKey();
cipher = Cipher.getInstance("AES");
System.out.println("Enter the plain text:");
String plainText = sc.nextLine( );
String encryptedText = encrypt(plainText, secretKey);
System.out.println("Secret Key:"+secretKey);
System.out.println("Encrypted Text: " + encryptedText);
}}
public static String encrypt(String plainText, SecretKey secretKey) throws Exception {
byte[] plainTextByte = plainText.getBytes();
cipher.init(Cipher.ENCRYPT_MODE, secretKey);
byte[] encryptedByte = cipher.doFinal(plainTextByte);
Base64.Encoder encoder = Base64.getEncoder();
String encryptedText = encoder.encodeToString(encryptedByte);
return encryptedText;
}}
```

Output:



```
D:\Academics\ISS\project>java AES.java
Enter the plain text:
VIT Chennai
Secret Key:javax.crypto.spec.SecretKeySpec@172ba
Encrypted Text: VQuM4GkCsB5tSJgHaSnXWg==
```

BLOWFISH: Blowfish has a 64-bit block size and a variable key length from 32 bits up to 448 bits. It is a 16-round Feistel cipher and uses large key-dependent S-boxes. In structure it resembles CAST-128, which uses fixed S-boxes.

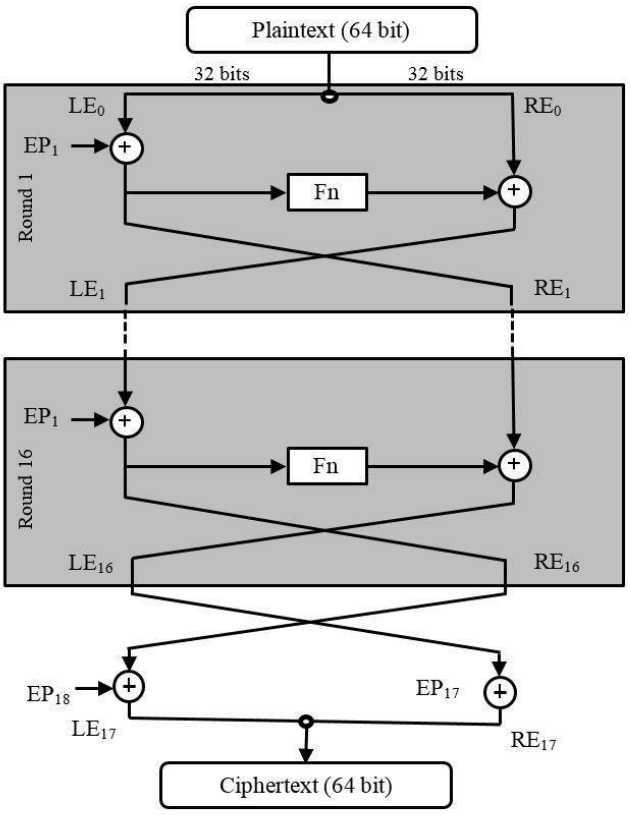


Figure 4: Blowfish Algorithm Structure

Each line represents 32 bits. There are five subkey-arrays: one 18-entry P-array (denoted as K in the diagram, to avoid confusion with the Plaintext) and four 256-entry S-boxes (S0, S1, S2 and S3). Every round *r* consists of 4 actions:

- Action 1 - XOR the left half (L) of the data with the *r*th P-array entry
- Action 2 - Use the XORed data as input for Blowfish's F-function
- Action 3 - XOR the F-function's output with the right half (R) of the data
- Action 4 - Swap L and R

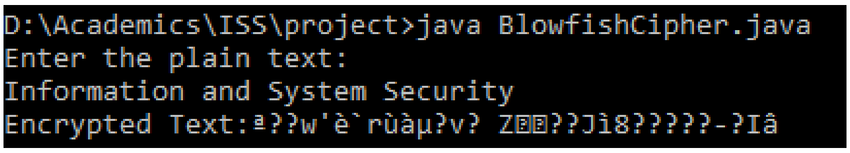
The F-function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The S-boxes accept 8-bit input and produce 32-bit output. The outputs are added modulo 2^{32} and XORed to produce the final 32-bit output. After the 16th round, undo the last swap, and XOR L with K18 and R with K17 (output whitening).

Blowfish's key schedule starts by initializing the P-array and S-boxes with values derived from the hexadecimal digits of pi, which contain no obvious pattern (see nothing up my sleeve number). The secret key is then, byte by byte, cycling the key if necessary, XORed with all the P-entries in order. A 64-bit all-zero block is then encrypted with the algorithm as it stands. The resultant ciphertext replaces P_1 and P_2. The same ciphertext is then encrypted again with the new subkeys, and the new ciphertext replaces P_3 and P_4. This continues, replacing the entire P-array and all the S-box entries. In all, the Blowfish encryption algorithm will run 521 times to generate all the subkeys - about 4KB of data is processed.

Code:

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.util.*;
public class BlowfishCipher {
public static void main(String[] args) throws Exception {
try (Scanner sc = new Scanner(System.in)) {
String message;
String keySrc = "test";
byte[] key = keySrc.getBytes();
SecretKeySpec ks = new SecretKeySpec(key,"Blowfish");
Cipher cipher = Cipher.getInstance("Blowfish");
System.out.println("Enter the plain text:");
message = sc.nextLine( );
cipher.init(Cipher.ENCRYPT_MODE, ks);
byte[] encrypted = cipher.doFinal(message.getBytes());
System.out.println("Encrypted Text:" + new String(encrypted));
}}}
```

Output:



```
D:\Academics\ISS\project>java BlowfishCipher.java
Enter the plain text:
Information and System Security
Encrypted Text:ª??w'è`rùàµ?v? Z▯▯??Jì8?????-?Iâ
```

Multiple Encryption:

Multiple Encryption is also known as cascade encryption, cascade ciphering, multiple ciphering. It is the process of encrypting an already encrypted message one or more times, either using the same or a different algorithm. Here, in this project, we take an input text and encrypt it with Caesar Cipher, the encrypted text from Caesar cipher becomes the plain text for the XOR cipher, and the output from this becomes the input for AES encryption and finally, this undergoes Blowfish encryption and gives us the final result.

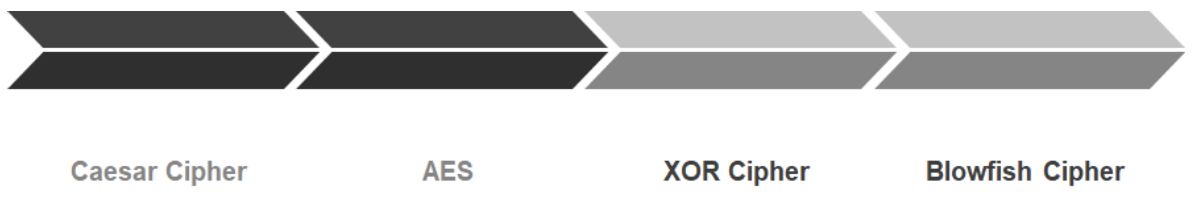


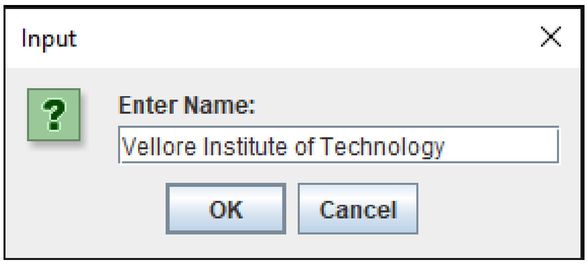
Figure 5 : Order of executing the Multiple Encryption

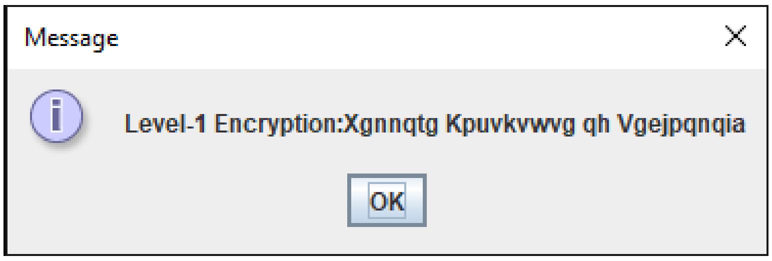
Code:

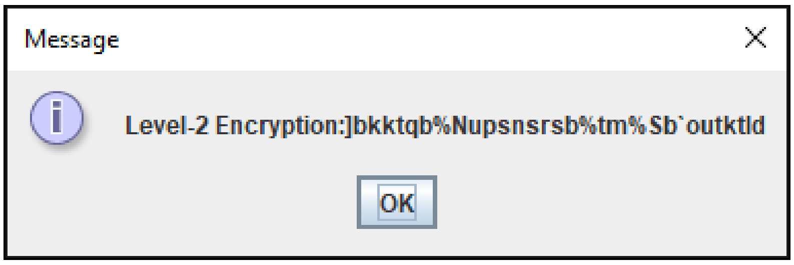
```
import java.util.*;
import javax.swing.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.io.*;
import java.util.Base64;
public class MultipleEncryption{
static Cipher cipher;
public static String CaesarCipher(String input, int k1) {
String result= new String();
char ch1[]=input.toCharArray();
for (int i=0; i<input.length(); i++) {
if(Character.isLetter(ch1[i])) {
if (Character.isUpperCase(input.charAt(i))) {
char ch = (char)(((int)input.charAt(i) + k1 - 65) % 26 + 65);
result=result+ch;
}
else {
char ch = (char)(((int)input.charAt(i) + k1 - 97) % 26 + 97);
result=result+ch;
}}
else if(ch1[i]==' ') {
char ch = ch1[i];
result=result+ch;
}}
```

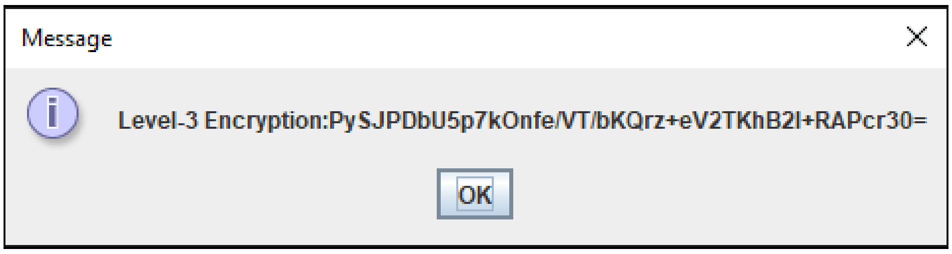
```
return result;
}
public static String AES(String l2, SecretKey k3) throws Exception{
byte[] plainTextByte = l2.getBytes();
cipher.init(Cipher.ENCRYPT_MODE, k3);
byte[] encryptedByte = cipher.doFinal(plainTextByte);
Base64.Encoder encoder = Base64.getEncoder();
String l3 = encoder.encodeToString(encryptedByte);
return l3;
}
public static String XOR(int k2,String l1){
String outputString = "";
int len = l1.length();
for (int i = 0; i < len; i++){
outputString = outputString + Character.toString((char) (l1.charAt(i) ^ k2));
}
return outputString;
}
public static void main(String args[]) throws Exception{
String input;
int k1,k2;
int max=26;
int limit=15;
String l1,l3,l4;
try (Scanner sc= new Scanner(System.in)){
input = JOptionPane.showInputDialog("Enter Name:");
Random rand = new Random();
k1 = rand.nextInt(max);
System.out.println("Level-1 key:"+k1);
l1 = CaesarCipher(input,k1);
JOptionPane.showMessageDialog(null,"Level-1 Encryption:"+l1);
k2 = rand.nextInt(limit);
System.out.println("Level-2 Key:"+k2);
String l2 = XOR(k2,l1);
JOptionPane.showMessageDialog(null,"Level-2 Encryption:"+l2);
KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
keyGenerator.init(128);
SecretKey k3 = keyGenerator.generateKey();
cipher = Cipher.getInstance("AES");
System.out.println("Level-3 Key:"+cipher);
l3 = AES(l2,k3);
JOptionPane.showMessageDialog(null,"Level-3 Encryption:"+l3);
String keySrc = "cipher";
byte[] k4 = keySrc.getBytes();
SecretKeySpec k44 = new SecretKeySpec(k4,"Blowfish");
Cipher cipher = Cipher.getInstance("Blowfish");
cipher.init(Cipher.ENCRYPT_MODE, k44);
byte[] encrypted = cipher.doFinal(l3.getBytes());
System.out.println("Level-4 Key:"+k4);
l4 = new String(encrypted);
JOptionPane.showMessageDialog(null,"Level-4 Encryption:"+l4);
}}}
```

Output:

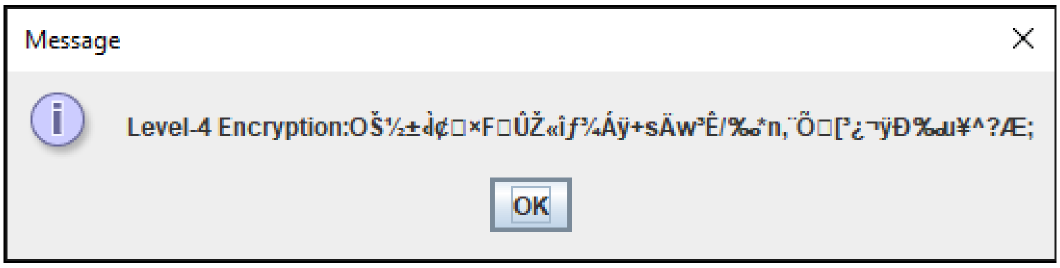



Level-1 Encryption:Xgnnqtg Kpuvkvwvg qh Vgejpqnqia

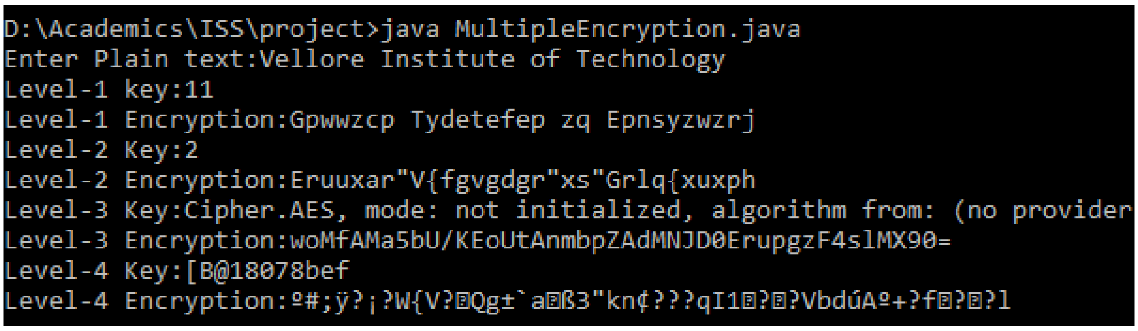

Level-2 Encryption:]bkktqb%Nupsnsrsb%tm%Sb`outktld



```
Level-3 Encryption:PySJPDbU5p7kOnfe/VT/bKQrz+eV2TKhB2l+RAPcr30=
```



```
Level-4 Encryption:OŠ½±ì¢□×F□ÛŽ«îƒ¼Áÿ+sÄw³Ê/‰*n,¨Õ□[³¿¬ÿÐ%ɑ¥^?Æ;
```



```
D:\Academics\ISS\project>java MultipleEncryption.java
Enter Plain text:Vellore Institute of Technology
Level-1 key:11
Level-1 Encryption:Gpwwzcp Tydetefep zq Epnsyzwzrj
Level-2 Key:2
Level-2 Encryption:Eruuxar"V{fgvgdgr"xs"Grlq{xuxph
Level-3 Key:Cipher.AES, mode: not initialized, algorithm from: (no provider
Level-3 Encryption:woMfAMa5bU/KEoUtAnmbpZAdMNJD0ErupgzF4slMX90=
Level-4 Key:[B@18078bef
Level-4 Encryption:º#;ÿ?¡?W{V?□Qg±`a□ß3"kn¢???qI1□?□?VbdúAº+?f□?□?l
```

COMAPARITIVE STUDY:

Multiple Encryption provides more provable security if you use independent keys for each level of encryption. Using the same key for all ciphers, allows easy brute force attacks. This also makes some sense in ridiculously high-security high-paranoia long term storage situations. The attacking gets easier for single level encryption whereas in multilevel encryption as the data gets encrypted several numbers of times it becomes difficult to attack. As the number of levels in encryption increases, more secure the data is. The Multiple Encryption provides better security than the existing single level encryption. It is laden with features that improve the security of an encrypting process. Multiple encryption allows for the encryption of an already-encrypted message one or more times using the same or different algorithm. It is a technique which supports wrapping a plaintext message in various crypto-wrappers in order to hide it more from intruders, and consequently, make it more secure. This new algorithm formulated for the Multiple or the Multi-level encryption wraps a plaintext message in four levels of crypto algorithms in order to allow for difficulty in breaking the cipher. However, the newly-developed algorithm uses automatically generated keys for all the four algorithms by random selection and Crypto Secret Key Generation methods. Hence, it employs 4 different keys, one for each encryption phase. This is also a property that gives the new algorithm an edge over the existing single layer encryptions.

RISK ANALYSIS:

- The size constraint of the input text is a huge point to keep in mind. This algorithm encrypts only the smaller size input texts entered in the input area.
- The order of encryption is another risk that has to be overcome. The order followed in this algorithm is Caesar Cipher, XOR Cipher, AES Encryption and the Blowfish Cipher respectively. If the order is slightly rearranged, then it might lead to an error.

CONCLUSION:

This project enhanced the different encryption algorithm with multiple encryption process and varying encryption key. Here the random key generation concept is used. The algorithm is tested on plain text and found more secure than the single level encryption algorithms. However, the new enhancement method can encrypt text through four different algorithms. The method of multiple encryption has been clearly shown to be more secure than any single level encryption.

REFERENCES:

- Dodis, Y. and Katz, J., 2005, February. Chosen-ciphertext security of multiple encryption. In *Theory of Cryptography Conference* (pp. 188-209). Springer, Berlin, Heidelberg.
- Dahl, U., Anonymity Protection in Sweden AB, 2001. *Data security system for a database having multiple encryption levels applicable on a data element value level*. U.S. Patent 6,321,201.
- Mohammed, B.B., 2013. Automatic key generation of Caesar Cipher. *Int. J. Eng. Trends Technol*, *6*(6), pp.2231-5381.
- Schneier, B., 1993, December. Description of a new variable-length key, 64-bit block cipher (Blowfish). In *International Workshop on Fast Software Encryption* (pp. 191-204). Springer, Berlin, Heidelberg.
- Mathur, N. and Bansode, R., 2016. AES based text encryption using 12 rounds with dynamic key selection. *Procedia Computer Science*, *79*, pp.1036-1043.