VIT®

Vellore Institute of Technology

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

School of Information Technology and Engineering

ENCRYPTION OF STORAGE DATA

A project submitted
in partial fulfillment of the requirements for the degree of
B.Tech. (Information Technology)

By

NADELLA VENKATA GAGAN ROHITH	(20BIT0025)
FANINDRA NAYAK	(20BIT0344)
DUDDU BALA GURU VENAKTA ARJUN	(20BIT0347)
IMMANI SAI BHARGAV	(20BIT0027)
TBS VINEETH	(20BIT0063)

Course Instructor

Dr. Selva Rani B

Associate Professor

Nov 2022

UNDERTAKING

This is to declare that the project entitled “Project Title” is an original work done by undersigned, in partial fulfillment of the requirements for the degree *B.Tech. (Information Technology)* at School of Information Technology and Engineering, VIT, Vellore.

All the analysis, design and system development have been accomplished by the undersigned.

NADELLA VENKATA GAGAN ROHITH	(20BIT0025)
FANINDRA NAYAK	(20BIT0344)
DUDDU BALA GURU VENAKTA ARJUN	(20BIT0347)
IMMANI SAI BHARGAV	(20BIT0027)
TBS VINEETH	(20BIT0063)

CONTENT

ABSTRACT

Cryptography is a way to encipher data. Securing data across communication channels is highly vital and it has to be safeguarded. At present, the organizations keep records of data on a cloud database. Although the cloud database enhances the efficiency of usage, it also poses a challenge to the secure storage of data.

We propose a hybrid algorithm using various modern and classical encryption techniques which will be used to encrypt data so we can safely store the data across various channels, the symmetric keys which will be used to decrypt the encrypted text.

The plaintext is converted into ciphertext, or unreadable data, using an encryption key and a specified encryption technique. Even if hackers manage to circumvent the system security measures, they won't be able to access the data because the scrambled information can only be unlocked with the associated encryption key.

In order to secure a wide range of information technology (IT) assets, encryption is crucial. It makes sure that the message's contents are encoded for confidentiality. Authentication verifies the source of message. And integrity demonstrates that a message's contents have not been altered after it was transmitted. Senders cannot claim they did not send the encrypted communication thanks to non-repudiation.

One of the commonly used data encryptions now a days is Secure Sockets Layer (SSL) which is used by websites, a kind of data encryption, to safeguard sensitive user information. It stops hackers from gaining access to private user information that is sent to and from the website. The URLs of websites that have incorporated SSL certificates display the padlock sign and use "https" rather than "http" for their link addresses.

What we are planning to develop secures the data which is stored at any facilities in cases when the data is compromised to unauthorized parties encryption makes it hard for attackers to understand or make use of any data they obtain.

1. INTRODUCTION

Encryption is an ancient tactic used in wars for sending secret message from one place to another, either physically or through a carrier. One of the oldest documented encryptions is used in Cira 600 BC The ancient Spartans used a device called a scytale to send secret messages during battle. This device consists of a leather strap wrapped around a wooden rod The letters on the leather strip are meaningless when it's unwrapped, and only if the recipient has the correctly sized rod does the message make sense. But the information encryption was first document in Circa 60 BC by Julius Caesarn, he invents a substitution cypher that shifts characters by three places: A becomes D, B becomes E, etc. A simple and effective encoding method at that time also known as Ceaser Cipher. Similarly, many different ciphering and deciphering techniques were invented till now. And as traditional storage devices such as flash drives, hard disks and other kinds of physical storage devices are slowly becoming obsolete. The reason for this is that, on the business front, global expansion of companies require data to be shared amongst employees for collaborative working. On the user's personal usage front, many users nowadays have multiple devices, such as one or more mobile/cell phones, tabs, laptops, desktop PCs et cetera. Hence cloud storage provides a way to access one's personal data across all of one's personal devices. Hence more and more people are shifting towards the more convenient option of cloud for storing their data. The ability to access files from remote locations using just a stable internet connection gives cloud an edge over other storage options.

2. LITERATURE SURVEY

Despite the usage of various types of cryptographic algorithms provide high security to information on networks, but they are also having some drawbacks. To improve the strength of these algorithms, authors of [1] propose a new hybrid cryptographic algorithm in this paper. In this paper two cryptography techniques were used (DES, IDEA).

Authors of [2] have proposed a model to preserve the privacy by randomizing the original data, then apply generalization on randomized or modified data. This technique protects private data with better accuracy, also it can reconstruct original data and provide data with no information loss, makes usability of data. Data is not secure in the cloud because the unauthorized user can try to use of the private data. So, providing the data security it uses the different encryption method to protect the data. So that in the proposed study [3] it uses the multilevel encryption algorithm. In the multilevel encryption it combines two different algorithms for providing the better security. A similar approach was also followed by [4].

The paper [5] describes in which A network, there is no complete security solution to secure data and applications or services, but satisfactory risk management can reduce the stage of risks. In our work, we used various techniques like AES, RSA and DES for proposed achieving higher security a cloud. The experimental results show that the hybrid encryption algorithm has the advantages of fast encryption and decryption speed, high security, good processing ability for longer data, and can solve the data security problem in cloud database to a certain extent [6].

The article written by [7] introduces a detailed analysis of the cloud security problem. In this paper various existing approaches related to data encryption and message authentications are discussed. After study the existing approaches, issues and challenges are point out during data processing over the cloud. This paper [8] explains User requires many important information based on their location to perform their task like location-based navigation, location-based information, and many more. The user has to give their important information like user identity and location information to the provider that are personalized.

The algorithm proposed by authors and researchers have their own advantages and disadvantages. Purpose of this paper [9] is to do exhaustive survey to find out research gap and challenges for suggesting a feasible solution with a new approach and to set future research direction. The focus of paper written by Carroll M [10] is on mitigation for cloud computing security risks as a fundamental step towards ensuring secure cloud computing environments. The paper [11] surveys the works on cloud security issues, making a comprehensive review of the literature on the subject. It addresses several key topics, namely vulnerabilities, threats, and attacks, proposing a taxonomy for their classification.

In this paper [12] a survey of security issues in terms of security threats and their remediations was presented. The contribution aims at the analysis and categorization of working mechanisms of the main security issues and the possible solutions that exist in the literature. Enabling of data sharing capabilities was discussed briefly in [13]. The authors of [14] have discussed about the basic features of the cloud computing, security issues, threats and their solutions. Additionally, the paper describes several key topics related to the cloud, namely cloud architecture framework, service and deployment model, cloud technologies, cloud security concepts, threats, and attacks.

Different combinations of encryption algorithms were analysed [15], on the basis of different performance parameters to deduce a hybrid algorithm which can secure data more efficiently on cloud.

3. MODULES AND DESCRIPTIONS

Modules

1. Ceaser Cipher
2. Vigenère Cipher
3. AES
4. DES
5. 3DES
6. SHA512

Module Description

1. Ceaser Cipher

The Caesar cipher is named after Julius Caesar, who, according to Suetonius, used it with a shift of three (A becoming D when encrypting, and D becoming A when decrypting) to protect messages of military significance.

The formula of encryption is :

$$En(x) = (x + n) \bmod 26$$

The formula of decryption is :

$$Dn(x) = (xi - n) \bmod 26$$

If any case (Dn) value becomes negative (-ve), in this case, we will add 26 in the negative value.

Where,

E denotes the encryption

D denotes the decryption

x denotes the letters value

n denotes the key value (shift value)

2. Vigenère Cipher

In 1586 Blaise de Vigenère published a type of polyalphabetic cipher called an autokey cipher – because its key is based on the original plaintext – before the court of Henry III of France. The cipher now known as the Vigenère cipher, however, is that originally described by Giovan Battista Bellaso in his 1553 book La cifra del Sig. Giovan Battista Bellaso. In Vigenere the encryption and decryption are done using algebraic formula in this encryption (convert the letters (A-Z) into the numbers (0-25)).

The formula of encryption is:

$$Ei = (Pi + Ki) \bmod 26$$

The formula of decryption is:

$$Di = (Ei - Ki) \bmod 26$$

If any case (Di) value becomes negative (-ve), in this case, we will add 26 in the negative value.

Where,

E denotes the encryption.

D denotes the decryption.

P denotes the plaintext.

K denotes the key.

3. AES

As a replacement for the Digital Encryption Standard algorithm, the AES(Advanced encryption Standard) algorithm was developed by the US National Institute of Standards and Technology. Two Belgian cryptographers, Joan Daemen and Vincent Rijmen, created the AES algorithm in 2001. As a symmetric-key algorithm, the AES algorithm uses the same key for both encryption and decryption.

The AES algorithm's standard key size is 128 bits long. Additionally, it supports keys with lengths of 192 and 256 bits. If we desire to apply an algorithm to a certain set of data, we must first define the block and key sizes. This algorithm accepts blocks of 128, 168, 224, 192, and 256 bits. The substitution-permutation network is the foundation for AES, which is effective in both hardware and software. AES does not employ a Feistel network, in contrast to DES, its predecessor As opposed to the Feistel cypher structure used by the DES algorithm, it operates on an SP network structure. Block cypher techniques use SP-networks, also known as substitution-permutation networks (SPN), which are a collection of connected mathematical processes. A round function, which accepts two inputs (a data block and a subkey) and produces one output the same size as the data block, is used in a Feistel network. The round function is applied to half of the data to be encrypted in each round, and the output is then XORed with the remaining half of the data. The number of rounds required to encrypt data using the AES variation of Rijndael, which has a fixed block size of 128 bits, depends on how long the key is. There are 10 rounds for a 128-bit key size, twelve rounds for a 192-bit key size, and fourteen rounds for a 256-bit key size.

If the data is not a multiple of the block size, padding is utilised. In this advanced encryption algorithm, padding has been used. The textual information may be of varying lengths. As a result, more padding is used. After reaching the necessary size, we apply padding before beginning the encryption process. Padding takes many distinct forms. The last byte represents the number of zeros, and there is padding with zeros. As further padding is added, the same byte is added each time until the required size is reached. Additional padding options include using random or null characters.

Procedures to be followed

There are few steps to be followed by every block after encrypting the individual blocks, it joins them together to form the final ciphertext.

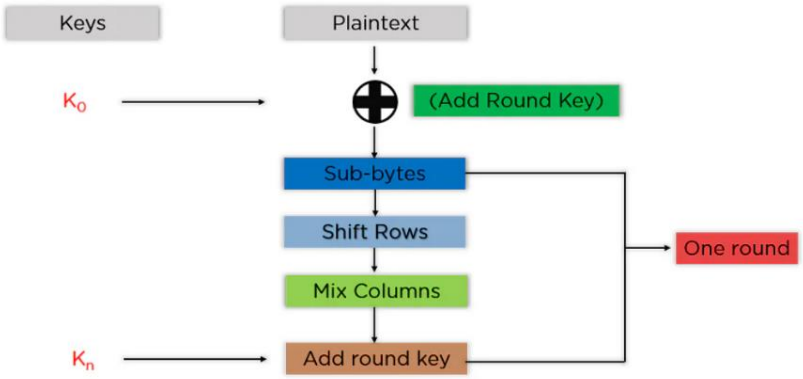

Figure 1 AES Algorithm

Sub-Bytes: In this phase, the state array's bytes are split into two equal portions and converted to hexadecimal. These components—rows and columns—are mapped using an S-Box substitution box to create new values for the final state array.

Figure 2 Sub-Bytes

Shift Rows: The row elements are switched around. There is a first row skip. The items in the second row are moved one position to the left. Additionally, it moves the third row's elements two successive locations to the left and moves the last row by three positions to the left.

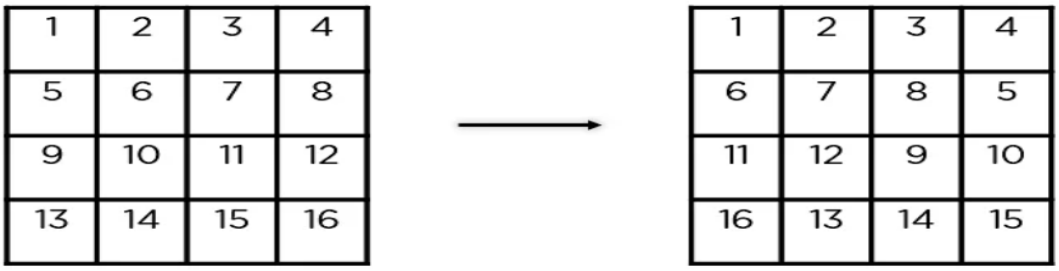

Figure 3 Shift Rows

Mix Columns: It multiplies a constant matrix with each column in the state array to get a new column for the subsequent state array. Once all the columns are multiplied with the same constant matrix, you get your state array for the next step. This particular step is not to be done in the last round.

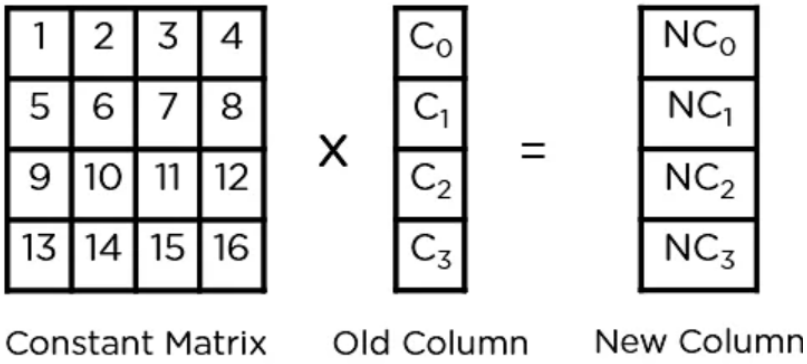

Figure 4 Mix Columns

Add Round Key: The state array obtained in the previous step is XORed with the appropriate key for the round. The resultant state array becomes the ciphertext for the particular block if this is the final round; otherwise, it serves as the new state array input for the following round.

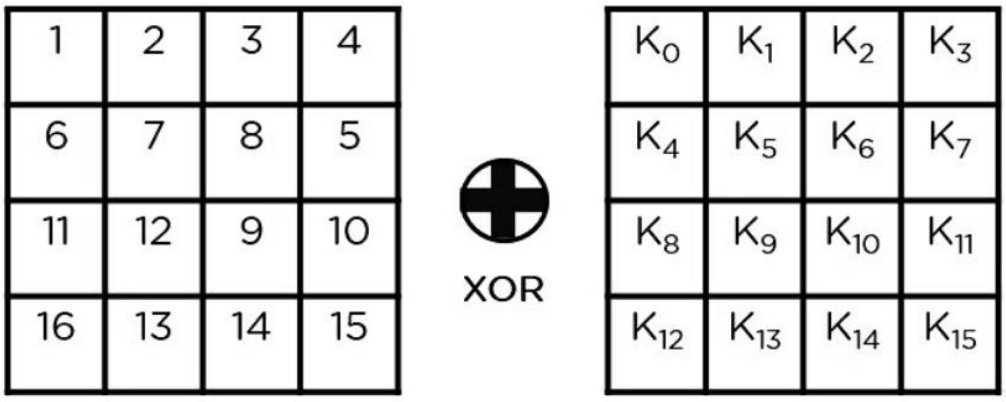

Figure 5 Add Round Key

4. DES

Data Encryption Standard is what it stands for. The DES algorithm can be cracked using certain devices. A key with a 56-bit size is used by the DES algorithm. The DES creates a block of 64-bit cypher text using this key after receiving a block of 64-bit plain text as input.

Each stage of the DES process, which consists of numerous phases, is referred to as a round. The amount of rounds changes depending on the size of the key being utilised. A 128-bit key, for instance, needs 10 rounds, a 192-bit key, 12 rounds, and so on.

A team from IBM developed the DES (Data Encryption Standard) algorithm, a symmetric-key block cypher that was later accepted by the National Institute of Standards and Technology (NIST). Using 48-bit keys, the technique transforms ordinary text, which is presented in 64-bit blocks, into ciphertext.

It uses the same key to encrypt and decrypt the data because it is a symmetric-key method. If the algorithm were asymmetrical, the encryption and decryption keys would be different.

Horst Feistel, an IBM cryptography specialist, created the LUCIFER block cypher in 1971, on which DES is based. The 16 rounds of the Feistel structure used by DES each have a unique key.

In November 1976, DES was certified as the government encryption standard, and it was later confirmed as the standard in 1983, 1988, and 1999.

The Advanced Encryption Standard (AES) supplanted the DES encryption algorithm as the acknowledged standard in 2002, ending the dominance of DES after a public search for a replacement. In May 2005, the NIST formally rescinded FIPS 46-3 (the 1999 reaffirmation), while Triple DES (3DES) is still permitted for sensitive government data till 2030.

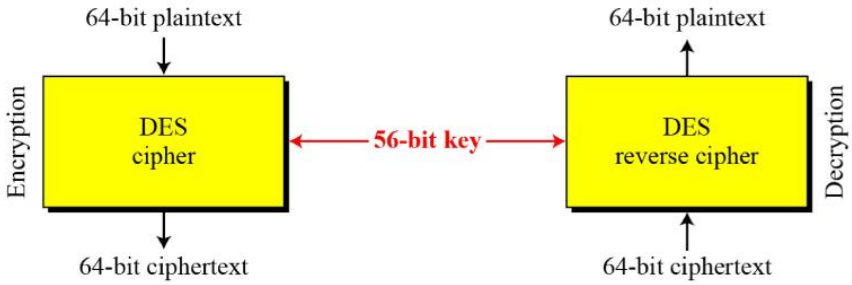

Figure 6 Overall DES

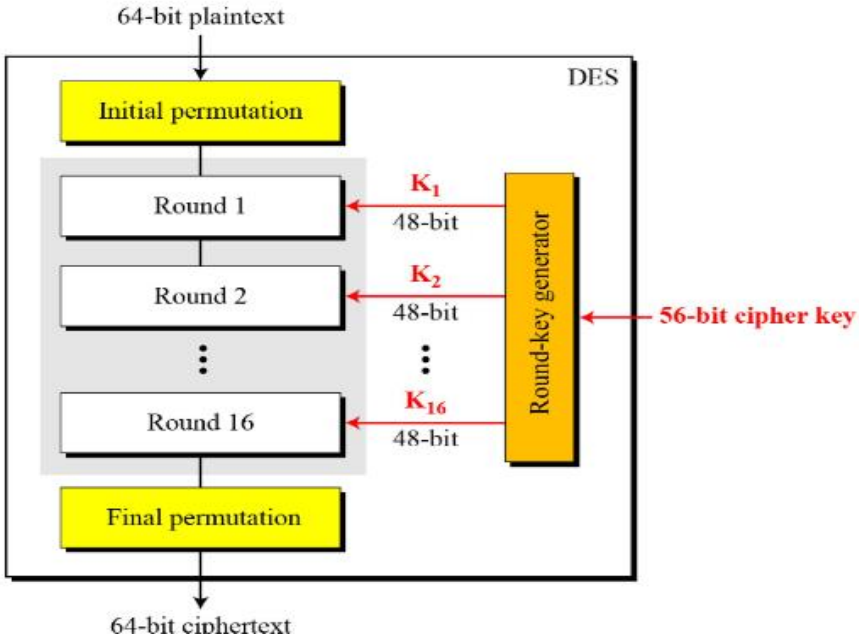

Figure 7 Expanded DES

Since DES is based on the Feistel Cipher, all that is required to specify DES is −

Round function

Key schedule

Any additional processing − Initial and final permutation

Initial and Final Permutation

The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES. The initial and final permutations are shown as follows –

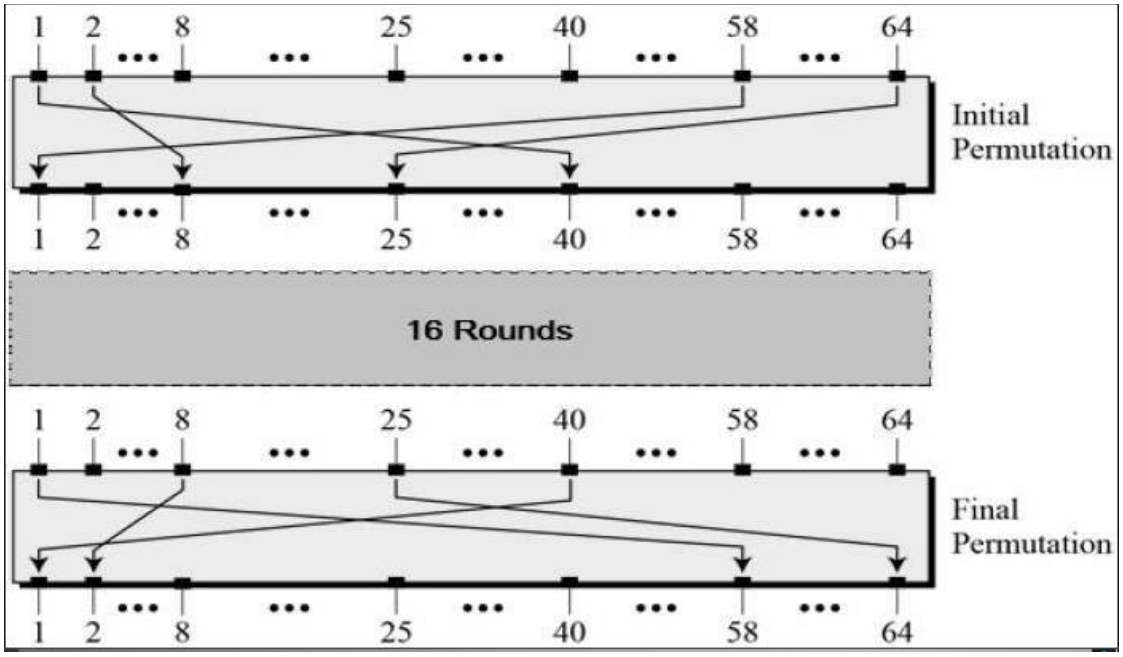

Figure 8 Initial and Final Permutation

Round Function

The heart of this cipher is the DES function, f. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.

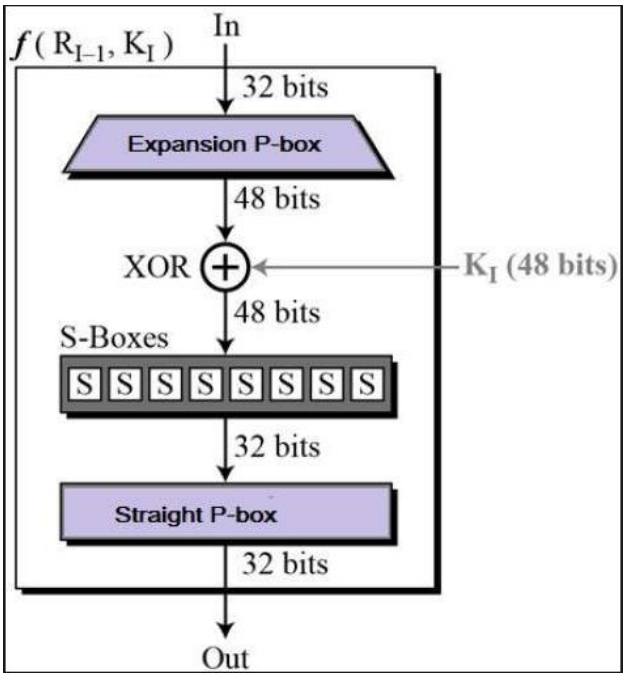

Figure 9 Round Function

Expansion Permutation Box

Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration –

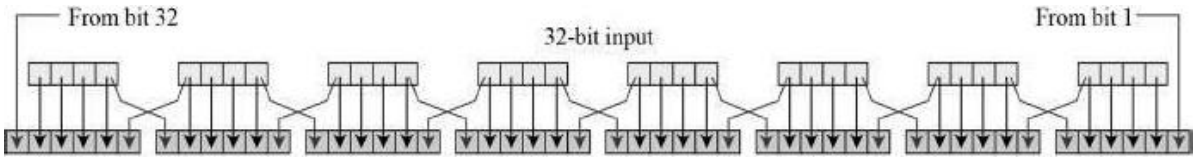

Figure 10 Expansion permutation box

The graphically depicted permutation logic is generally described as table in DES specification illustrated as shown –

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

Figure 11 Permutation logic box

XOR (Whitener)

After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.

Substitution Boxes. − The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. Refer the following illustration –

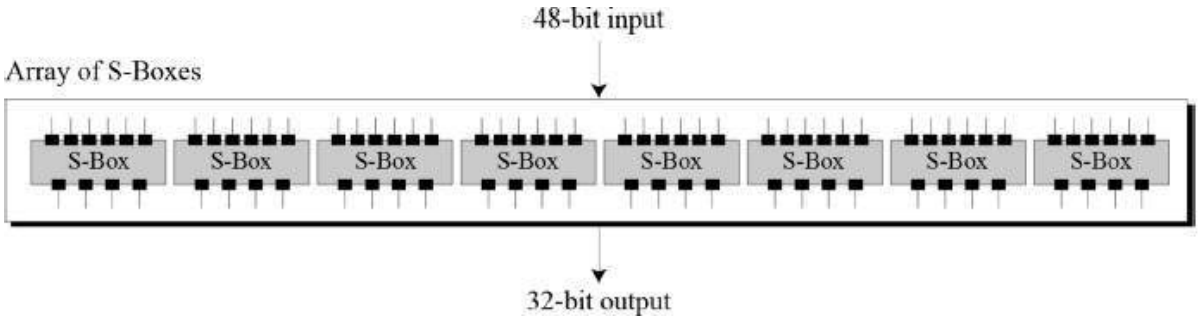

Figure 12 48 bit output to 32 bit conversion

The S-box rule is illustrated below –

There are a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32 bit section.

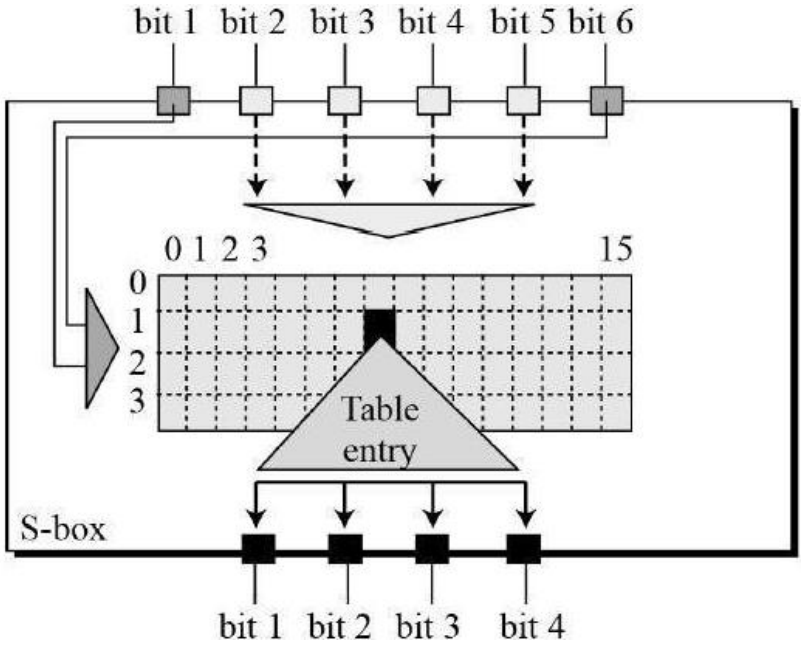

Figure 13 working of an S-box

Straight Permutation

The 32 bit output of S-boxes is then subjected to the straight permutation with rule shown in the following illustration:

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Figure 14 Straight Permutation

Depiction of One Round in DES

Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. The process of key generation is depicted in the Figure 16 Key Generation

The logic for Parity drop, shifting, and Compression P-box is given in the DES description.

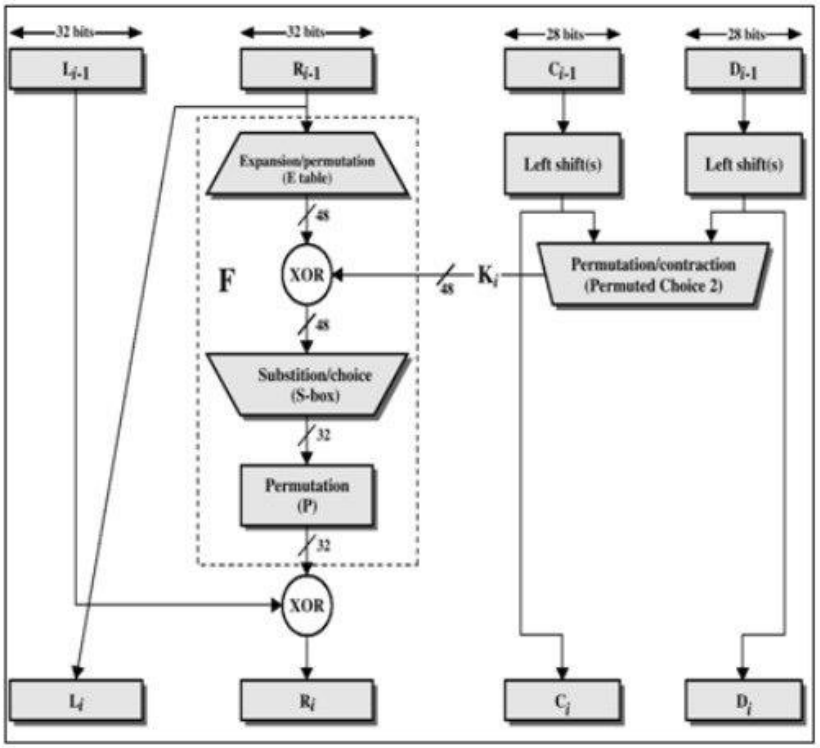

Figure 15 Illustration of a single round in DES

DES Analysis

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

Avalanche effect − A small change in plaintext results in the very great change in the ciphertext.

Completeness − Each bit of ciphertext depends on many bits of plaintext.

During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.

DES has proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

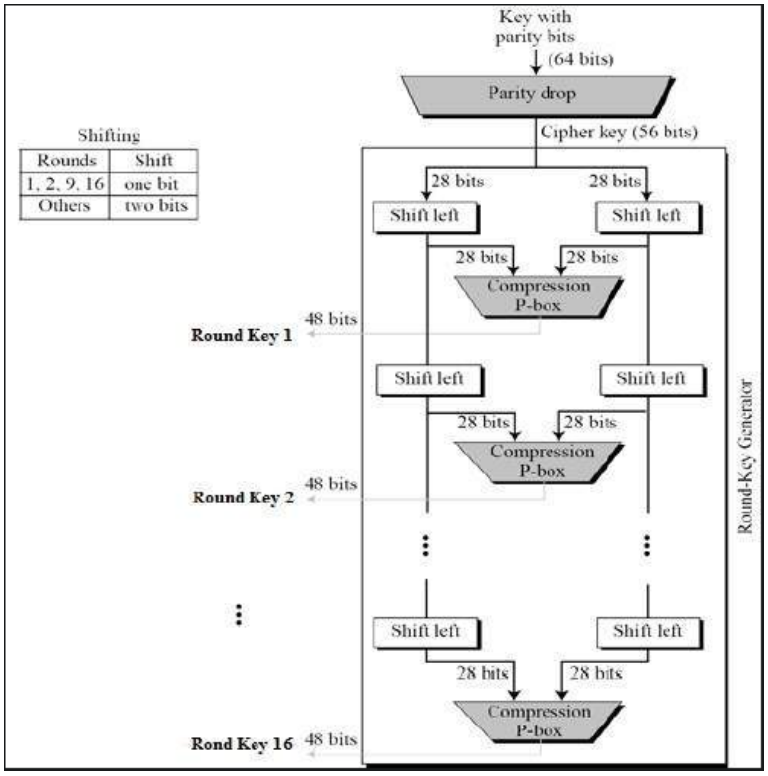

Figure 16 Key Generation

5. 3DES

Triple DES is a encryption technique which uses three instance of DES on same plain text. It uses there different types of key choosing technique in first all used keys are different and in second two keys are same and one is different and in third all keys are same.

Triple DES is also vulnerable to meet-in-the middle attack because of which it give total security level of 2^112 instead of using 168 bit of key. The block collision attack can also be done because of short block size and using same key to encrypt large size of text. It is also vulnerable to sweet32 attack.

The encryption algorithm is:

$$\text{ciphertext} = E_{K3}(D_{K2}(E_{K1}(\text{plaintext}))).$$

That is, DES encrypt with K1, DES *decrypt* with K2, then DES encrypt with K3.

Decryption is the reverse:

$$\text{plaintext} = D_{K1}(E_{K2}(D_{K3}(\text{ciphertext}))).$$

That is, decrypt with K3, encrypt with K2, then decrypt with K1.

Due to this design of Triple DES as an encrypt–decrypt–encrypt process, it is possible to use a 3TDES (hardware) implementation for single DES by setting K_1, K_2, and K_3 to be the same value. This provides backwards compatibility with DES. Second variant of Triple DES (2TDES) is identical to 3TDES except that K3is replaced by K1. In other words, user encrypt plaintext blocks with key K1, then decrypt with key K2, and finally encrypt with K1 again. Therefore, 2TDES has a key length of 112 bits.Triple DES systems are significantly more secure than single DES, but these are clearly a much slower process than encryption using single DES.

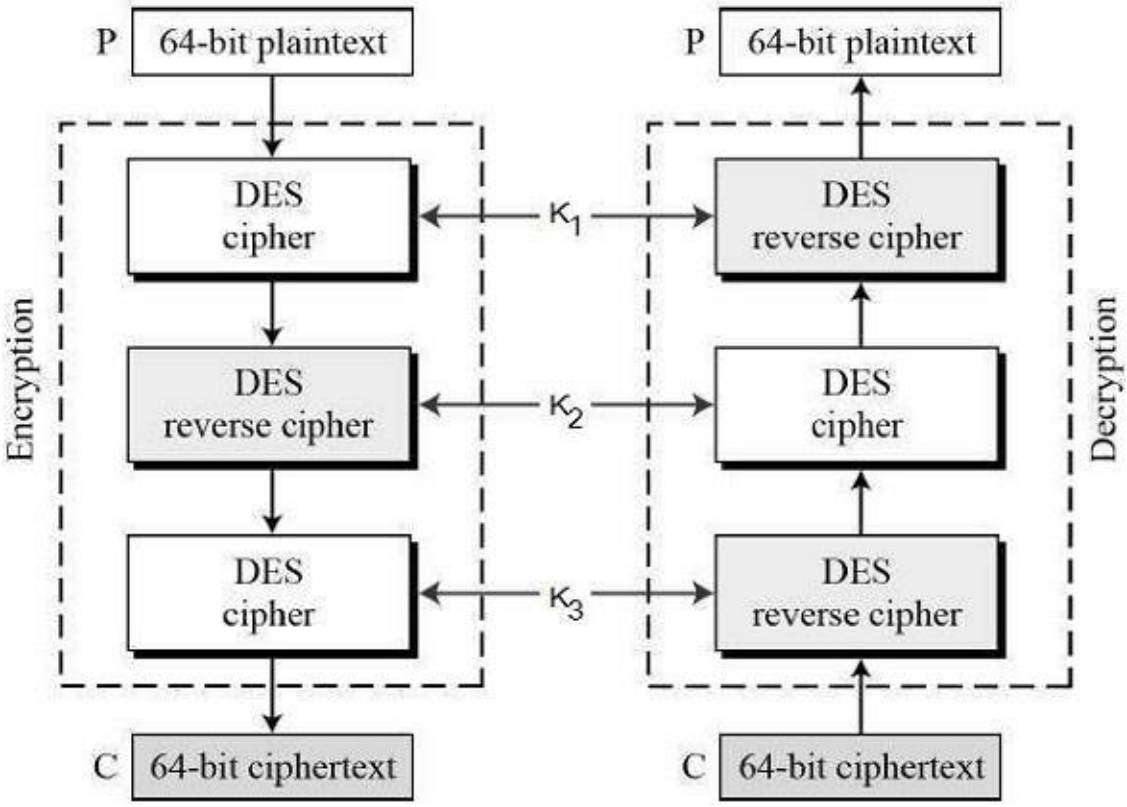

Figure 17 Working of Triple DES

6. SHA512

SHA-512 is a hashing algorithm that performs a hashing function on some data given to it.

SHA-512 does its work in a few stages. These stages go as follows:

1. Input formatting
2. Hash buffer initialization
3. Message Processing
4. Output

Input Formatting:

SHA-512 can't actually hash a message input of any size, i.e. it has an input size limit. This limit is imposed by its very structure as you may see further on. The entire formatted mesage has basically three parts: the original message, padding bits, size of original message. And this should all have a combined size of a whole multiple of 1024 bits. This is because the formatted message will be processed as blocks of 1024 bits each, so each bock should have 1024 bits to work with.

Figure 18 Original Message Length

Padding bits

The input message is taken and some padding bits are appended to it in order to get it to the desired length. The bits that are used for padding are simply '0' bits with a leading '1' (100000…000). Also, according to the algorithm, padding *needs* to be done, even if it is by one bit. So a single padding bit would only be a '1'.

The total size should be equal to 128 bits short of a multiple of 1024 since the goal is to have the formatted message size as a multiple of 1024 bits (N x 1024).

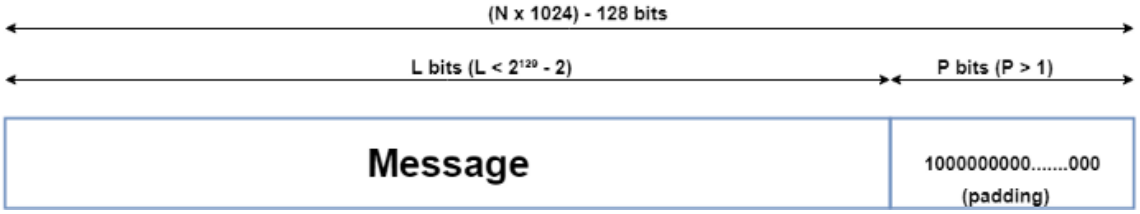

Figure 19 Padding Bits with Original Message

Padding size

After this, the size of the original message given to the algorithm is appended. This size value needs to be represented in 128 bits and is the only reason that the SHA-512 has a limitation for its input message.

Since the size of the original message needs to be represented in 128 bits and the largest number that can be represented using 128 bits is (2^{128}-1), the message size can be at most (2^{128}-1) bits; and also taking into consideration the necessary single padding bit, the maximum size for the original message would then be (2^{128}-2). Even though this limit exists, it doesn't actually cause a problem since the actual limit is so high (2^{128}-2 = 340,282,366,920,938,463,463,374,607,431,768,211,454 bits).

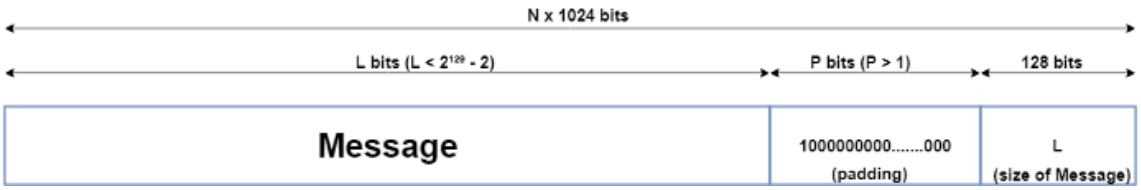

Figure 20 Padding bits+ length of message + original message

Now that the padding bits and the size of the message have been appended, we are left with the completely formatted input for the SHA-512 algorithm.

Figure 21 Formatted input

Hash buffer initialization:

The algorithm works in a way where it processes each block of 1024 bits from the message using the result from the previous block. Now, this poses a problem for the first 1024 bit block which can't use the result from any previous processing. This problem can be solved by using a default value to be used for the first block in order to start off the process. (Have a look at the second-last diagram).

Since each intermediate result needs to be used in processing the next block, it needs to be stored somewhere for later use. This would be done by the *hash buffer*, this would also then hold the final hash digest of the entire processing phase of SHA-512 as the last of these 'intermediate' results.

So, the default values used for starting off the chain processing of each 1024 bit block are also stored into the hash buffer at the start of processing. The actual value used is of little consequence, but for those interested, the values used are obtained by taking the first 64 bits of the fractional parts of the square roots of the first 8 prime numbers (2,3,5,7,11,13,17,19). These values are called the Initial Vectors (IV).

Why 8 prime numbers instead of 9? Because the hash buffer actually consists of 8 subparts (registers) for storing them.

Figure 22 Hash Buffers representation

Initialization Vector

a = 0x6A09E667F3BCC908 b = 0xBB67AE8584CAA73B

c = 0x3C6EF372FE94F82B d = 0xA54FF53A5F1D36F1

e = 0x510E527FADE682D1 f = 0x9B05688C2B3E6C1F

g = 0x1F83D9ABFB41BD6B h = 0x5BE0CD19137E2179

Figure 23 Initialization vectors

Message Processing:

Message processing is done upon the formatted input by taking one block of 1024 bits at a time. The actual processing takes place by using two things: The 1024 bit block, and the result from the previous processing.

This part of the SHA-512 algorithm consists of several 'Rounds' and an addition operation.

So, the Message block (1024 bit) is expanded out into 'Words' using a 'message sequencer'. Eighty Words to be precise, each of them having a size of 64 bits.

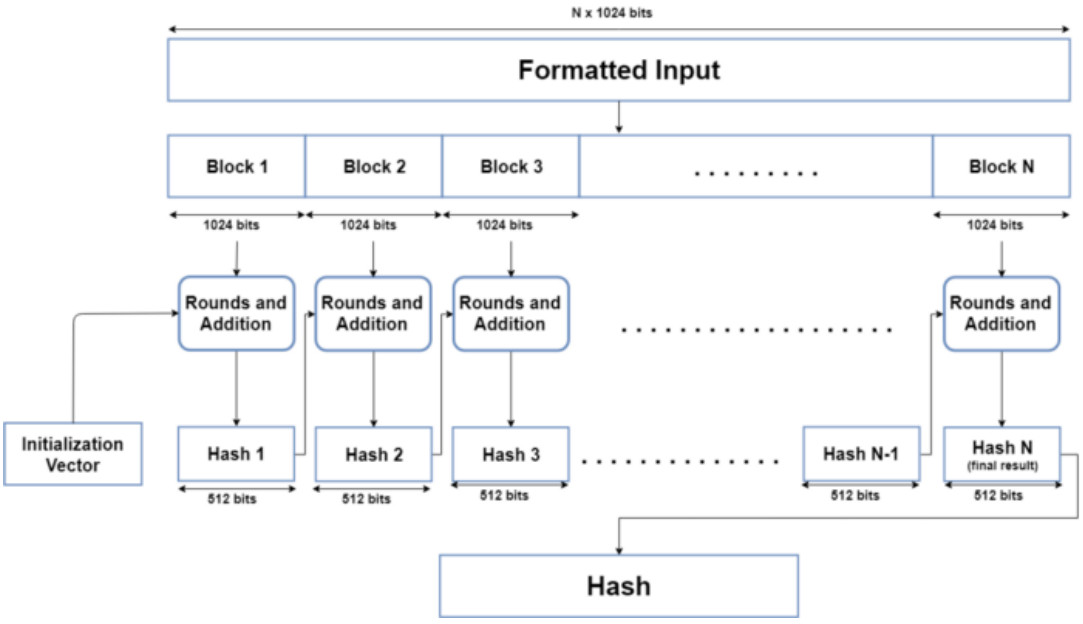

Figure 24 Message Processing

Rounds

The main part of the message processing phase may be considered to be the Rounds. Each round takes 3 things: one Word, the output of the previous Round, and a SHA-512 constant. The first Round doesn't have a previous Round whose output it can use, so it uses the final output from the previous message processing phase for the previous block of 1024 bits. For the first Round of the first block (1024 bits) of the formatted input, the Initial Vector (IV) is used.

SHA-512 constants are predetermined values, each of whom is used for each Round in the message processing phase. Again, these aren't very important, but for those interested, they are the first 64 bits from the fractional part of the cube roots of the first 80 prime numbers.Why 80? Because there are 80 Rounds and each of them needs one of these constants.

Once the Round function takes these 3 things, it processes them and gives an output of 512 bits. This is repeated for 80 Rounds. After the 80th Round, its output is simply added to the result of the previous message processing phase to get the final result for this iteration of message processing.

Output:

After every block of 1024 bits goes through the message processing phase, i.e. the last iteration of the phase, we get the final 512 bit Hash value of our original message. So, the intermediate results are all used from each block for processing the next block. And when the final 1024 bit block has finished being processed, we have with us the final result of the SHA-512 algorithm for our original message.

Thus, we obtain the final hash value from our original message. The SHA-512 is part of a group of hashing algorithms that are very similar in how they work, called SHA-2. Algorithms such as SHA-256 and SHA-384 are a part of this group alongside SHA-512. SHA-256 is also used in the Bitcoin blockchain as the designated hash function.

That's a brief overview of how the SHA-512 hashing algorithm works.I intend to go into further detail about what makes the hash functions practically irreversible (one-way) and how this is helpful for digital security.

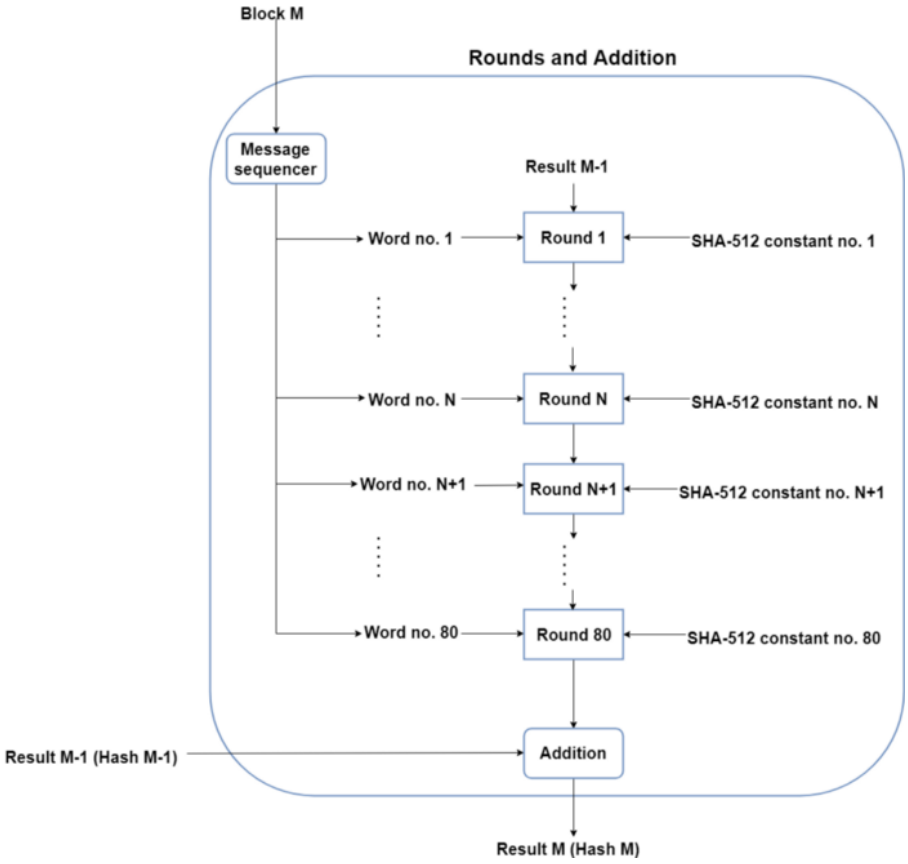

Figure 25 Round in SHA -512

4. Implementation

Coding for encrypting and decrypting the text

Project.py

```
from time import time

timestamp = int(time() * 1000)

from Ceaser import *
from Vigenère import *
from aes import *
from des import *
from sha import *
from Tripledes import *

from tkinter import *
from tkinter import filedialog

def func(PlainText,VignerCipherKey,c,d):

    # PlainText = "I wandered lonely as a cloud That floats on."
    # VignerCipherKey = "asdasdasdasdasdasdasdasdasdasdasdasdasdasdasdasdasd"
    # DesCipherKey = b'hello123'
    # AESCipherKey = b'Sixteen byte key'

    PlainText = PlainText
```

```
    VignerCipherKey = VignerCipherKey
    DesCipherKey = c.encode()
    AESCipherKey = d.encode()
    CeaserCipherKey = len(VignerCipherKey)//3

    def
enc(PlainText,CeaserCipherKey,AESCipherKey,VignerCipherKey,DesCipherKey):
        # Step 1 Ceaser Cipher encryption
        arr = []

        CeaserEncryptedText =  CeaserEncrypt(PlainText,CeaserCipherKey)
        arr.append("----------   ENCRYPTION  ----------")
        arr.append("\n")
        arr.append("Time  Stamp is :- ")
        arr.append(timestamp)
        arr.append("Plain  Text after Ceaser Cipher Encryption : \n")
        arr.append("\n")
        arr.append("Time  Stamp is :- ")
        arr.append(timestamp)
        arr.append("\n")
        arr.append(CeaserEncryptedText)

        # Step 2 AES encryption

AesEncrypttedText=AESEncryption(CeaserEncryptedText.encode(),AESCipherKey)
        arr.append("\n\n")
        arr.append("Encrypted Plain Text obatined after AES encryption : \n")
```

```
arr.append("\n")
arr.append("Time  Stamp is :- ")
arr.append(timestamp)
arr.append("\n")
strg=str(AesEncrypttedText[1])
arr.append(strg)
arr.append("\n\n")
arr.append("\n\n")
arr.append("The  sha for the aes encrypted text is")
shaValue = sha(AesEncrypttedText[1])
arr.append("\n")
arr.append("Time  Stamp is :- ")
arr.append(timestamp)
arr.append("\n")
strSha = str(shaValue)
arr.append(strSha)
arr.append("\n\n")
arr.append("\n\n")



# Step 3 for key Encryption using  Vigenere


dec=vigenere_encrypt(AESCipherKey.decode(),VignerCipherKey)
arr.append("\n\n")
arr.append("Key after Vigenere Encryption: \n")
arr.append("\n")
arr.append("Time  Stamp is :- ")
```

```
arr.append(timestamp)
arr.append("\n")
arr.append(dec)

# Step 4 for key Encryption using DES
timeStampDesBegning =int(time() * 1000)

dest =DesEncrryption(dec.encode(),DesCipherKey)
arr.append("\n\n")
arr.append("Key obtained from DES Encryption: \n")
arr.append("\n")
arr.append("\n")
arr.append("Time  Stamp is :- ")
arr.append(timestamp)
strg1=str(dest)
arr.append("\n")
arr.append(strg1)

timeStampDesEnd =int(time() * 1000)
timestampOfDes = timeStampDesEnd-timeStampDesBegning
arr.append("\n\n")
arr.append("Time  taken in millisecond  is ")
arr.append(timestampOfDes)
arr.append("\n\n")

# Triple  Des
```

```
        Tripdest = triipleDesEncode(dec)



        return dest,AesEncrypttedText,arr,Tripdest



    def
dec(AesEncrypttedText,CeaserCipherKey,DesCipherKey,dest,VignerCipherKey,Tripdest
):
        arrD = []
        arrD.append("----------   DECRYPTION  ----------\n")
        # Step 1 Decrypting received Plain text using AES


        AesDecrypttedText =
AESDEcryption(AesEncrypttedText[1],AESCipherKey,AesEncrypttedText[0])


        arrD.append("\n\n")
        arrD.append("Decrypting received Plain text using AES: \n")
        arrD.append("\n")
        arrD.append("Time Stamp is :- ")
        arrD.append(timestamp)
        arrD.append("\n")
        arrD.append(AesDecrypttedText)


        # Step 2 Decrypting output of AES with ceaser


        CeaserDecryptedText =
CeaserCipherDecrypt(AesDecrypttedText,CeaserCipherKey)
        arrD.append("\n\n")
```

```
arrD.append("Original Plain Text: \n")
arrD.append("\n")
arrD.append("Time Stamp is :- ")
arrD.append(timestamp)
arrD.append("\n")
arrD.append(CeaserDecryptedText)

# Step 3 for key Decrypt using DES

dest1 =DesDecryption(dest,DesCipherKey)
arrD.append("\n\n")
arrD.append("Decrypting Key using DES: \n")
arrD.append("\n")
arrD.append("Time Stamp is :- ")
arrD.append(timestamp)
arrD.append("\n")
strG =str(dest1)
arrD.append(strG)

DeTripDes =triipleDesDecode(Tripdest)

# Step 4 for key Decryption of output of DES with vigenere

dectxt=vigenere_decrypt(dest1.decode(),VignerCipherKey)
arrD.append("\n\n")
arrD.append("Secret Key: \n")
arrD.append("\n")
```

```
        arrD.append("Time Stamp is :- ")
        arrD.append(timestamp)
        arrD.append("\n")
        arrD.append(dectxt)

        return dest1,arrD


dest=enc(PlainText,CeaserCipherKey,AESCipherKey,VignerCipherKey,DesCipherKey)


deDest=dec(dest[1],CeaserCipherKey,DesCipherKey,dest[0],VignerCipherKey,dest[3])

    return dest[2],deDest[1]


def openFile():
    tf = filedialog.askopenfilename(
        initialdir="C:/Users/MainFrame/Desktop/",
        title="Open Text file",
        filetypes=(("Text Files", "*.txt"),)
        )
    pathh.insert(END, tf)
    tf = open(tf)  # or tf = open(tf, 'r')
    data = tf.read().splitlines()
    newDAta=func(data[0],data[1],data[2],data[3])
```

```
    # newData1 = newDAta[0].split(",")
    things_To_add = newDAta[0]+newDAta[1]

    stringing =""
    for i in range(len(things_To_add)):

        stringing=stringing+ str(things_To_add[i])
    stringingEnc =""
    for i in range(len(newDAta[0])):

        stringingEnc=stringingEnc+ str(newDAta[0][i])
    stringingDec =""
    for i in range(len(newDAta[1])):

        stringingDec=stringingDec+ str(newDAta[1][i])
    my_file = open("enc.txt","w+")
    my_file.write(stringingEnc)
    my_file = open("dec.txt","w+")
    my_file.write(stringingDec)
    txtarea.insert(END, stringingEnc)
    txtarea1.insert(END, stringingDec)
    tf.close()
    return data


def saveFile():
    tf = filedialog.asksaveasfile(
```

```
        mode='w',

        title ="Save file",
        defaultextension=".txt"
        )
    # tf.config(mode='w')

    # pathh.insert(END, tf)
    if tf is None:
        return

    data = str(txtarea.get(1.0, END))
    tf.write(data)

    tf.close()


ws = Tk()
ws.title("Enigma")
ws.geometry("1000x500")
ws['bg']='#fb0'
# adding frame
frame = Frame(ws)
frame.pack(pady=20)

# adding scrollbars
ver_sb = Scrollbar(frame, orient=VERTICAL )
```

```
ver_sb.pack(side=RIGHT, fill=BOTH)

hor_sb = Scrollbar(frame, orient=HORIZONTAL)
hor_sb.pack(side=BOTTOM, fill=BOTH)

# adding writing space
txtarea = Text(frame, width=40, height=20)
txtarea.pack(side=LEFT)
frame.pack(pady=20,padx=20)
# adding writing space
txtarea1 = Text(frame, width=40, height=20)
txtarea1.pack(side=LEFT)

# binding scrollbar with text area
txtarea.config(yscrollcommand=ver_sb.set)
ver_sb.config(command=txtarea.yview)

txtarea.config(xscrollcommand=hor_sb.set)
hor_sb.config(command=txtarea.xview)

pathh = Entry(ws)
pathh.pack(side=LEFT, expand=True, fill=X, padx=20)




Button(
   ws,
```

```
    text="Open File",
    command=openFile
    ).pack(side=RIGHT, expand=True, fill=X, padx=20)

Button(
    ws,
    text="Save File",
    command=saveFile
    ).pack(side=LEFT, expand=True, fill=X, padx=20)

ws.mainloop()
```

Ceaser.py

```
def CeaserEncrypt(message,key):
    message = message.upper()
    alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    result = ""

    for letter in message:
        if letter in alpha:
            letter_index = (alpha.find(letter) + key) % len(alpha)

            result = result + alpha[letter_index]
        else:
            result = result + letter

    return result
```

```
def CeaserCipherDecrypt(message,key):
    message = message.upper()
    alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    result = ""

    for letter in message:
        if letter in alpha:
            letter_index = (alpha.find(letter) - key) % len(alpha)

            result = result + alpha[letter_index]
        else:
            result = result + letter

    return result
```

Vigenere.py

```
def vigenere(
        text: str,
        key: str,

alphabet='0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstu
vwxyz. ',
        encrypt=True
):

    result = ''

    for i in range(len(text)):
```

```
        letter_n = alphabet.index(text[i])
        key_n = alphabet.index(key[i % len(key)])

        if encrypt:
            value = (letter_n + key_n) % len(alphabet)
        else:
            value = (letter_n - key_n) % len(alphabet)

        result += alphabet[value]

    return result


def vigenere_encrypt(text, key):
    return vigenere(text=text, key=key, encrypt=True)


def vigenere_decrypt(text, key):
    return vigenere(text=text, key=key, encrypt=False)
```

aes.py

```
from Crypto.Cipher import AES
def AESEncryption(text,key):
    cipher = AES.new(key, AES.MODE_EAX)
    ciphertext = cipher.encrypt(text)

    return cipher,ciphertext
def AESDEcryption(ctext,key,cipher):
```

```
    nonce = cipher.nonce
    cipher =  AES.new(key, AES.MODE_EAX, nonce)
    plaintext = cipher.decrypt(ctext)
    return plaintext.decode()
```

des.py

```
from Crypto.Cipher import DES

def pad(text):
    n = len(text) % 8
    return text + (b' ' * n)

def DesEncrryption(text,key):

    des = DES.new(key, DES.MODE_ECB)
    padded_text = pad(text)
    encrypted_text = des.encrypt(padded_text)
    return encrypted_text

def DesDecryption(cipher,key):
    des = DES.new(key, DES.MODE_ECB)
    descryptedText = des.decrypt(cipher)
    return descryptedText
```

sha.py

```
import hashlib
m = hashlib.sha512()
def sha(txt):
    m.update(txt)
```

```
    return m.digest()
```

Tripledes.py

```
from Crypto.Cipher import DES3
from Crypto.Random import get_random_bytes
while True:
    try:
        key = DES3.adjust_key_parity(get_random_bytes(24))
        break
    except ValueError:
        pass
def triipleDesEncode(txt):
    cipher = DES3.new(key, DES3.MODE_CFB)
    plaintext = txt.encode()
    msg = cipher.iv + cipher.encrypt(plaintext)
    return msg
def triipleDesDecode(msg):
    cipher = DES3.new(key, DES3.MODE_CFB)
    decryptmsg = cipher.iv + cipher.decrypt(msg)
    return decryptmsg[16:]
```

5. Results

Output GUI

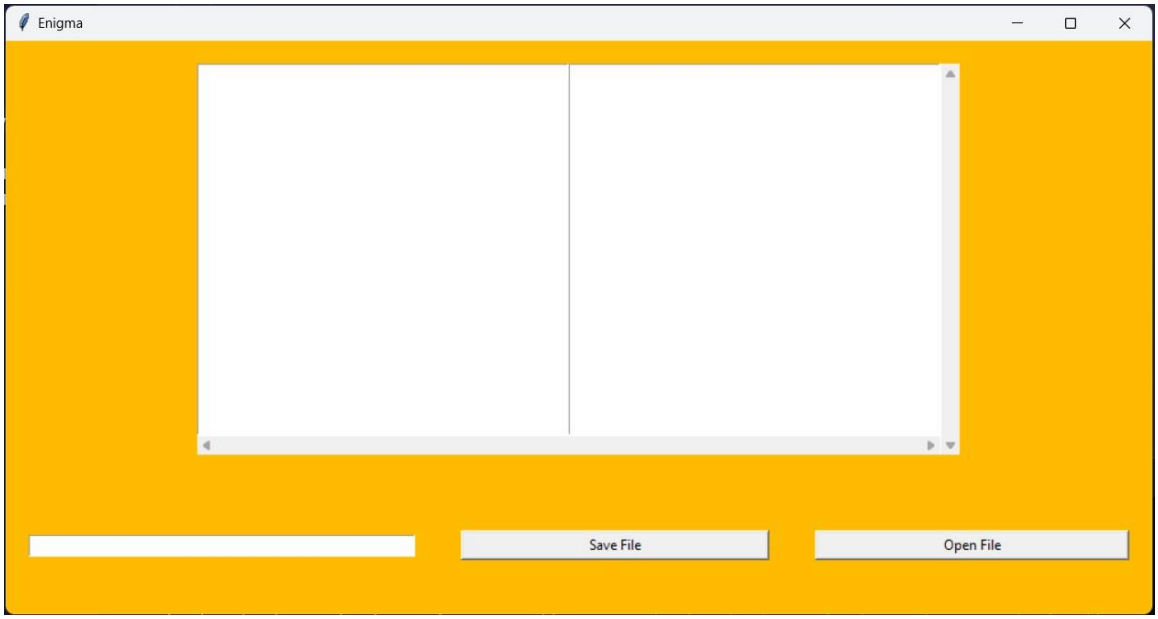

Figure 26 Blank GUI

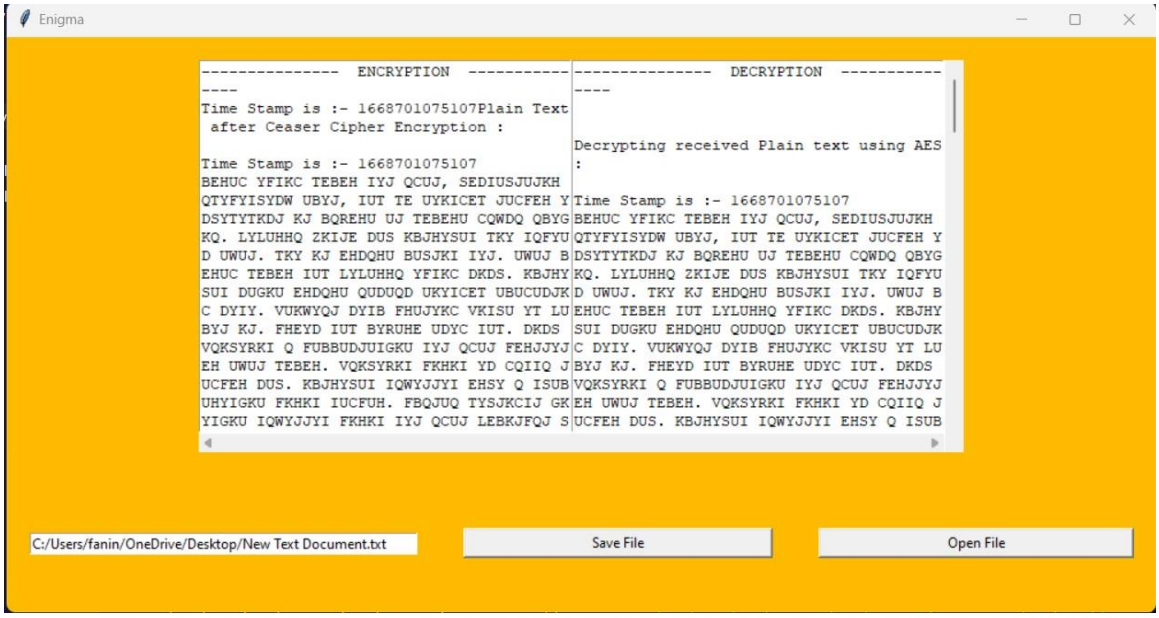

Figure 27 GUI with input and Output

Input Text File

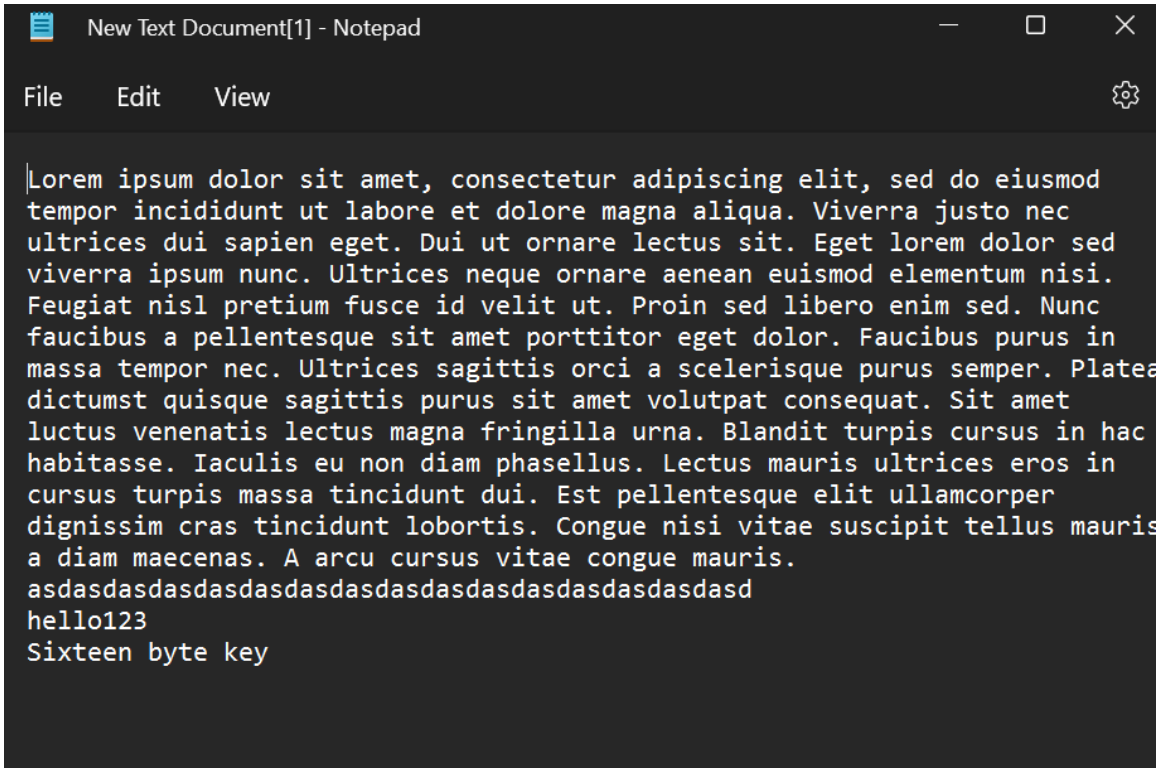

Figure 28 Input text file

Encrypted Text File

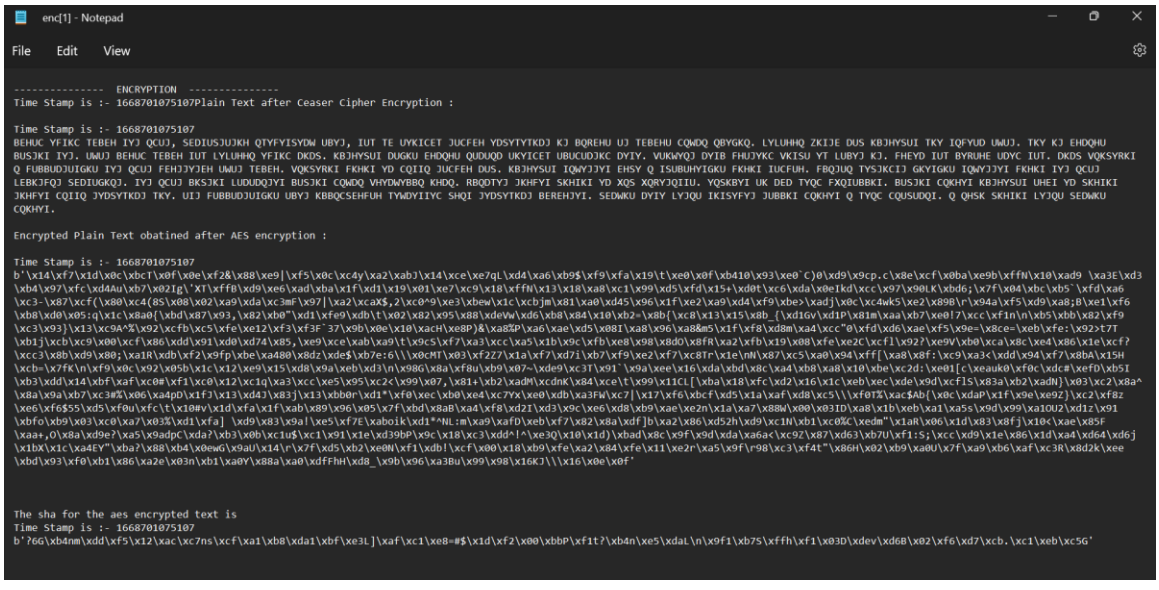

Figure 29 Encrypted text file

```
Key after Vigenere Encryption:

Time Stamp is :- 1668701075107
0YYRUFLrCWjFZaFW

Key obtained from DES Encryption:


Time Stamp is :- 1668701075107
b'9\xd4U\x07\x04i<\x9dP\x0f\xc2\x04\x9cN\xe8\x11'

Time taken in millisecond is 2
```

Figure 30 Encrypted text file (part 2)

Decrypted Text File

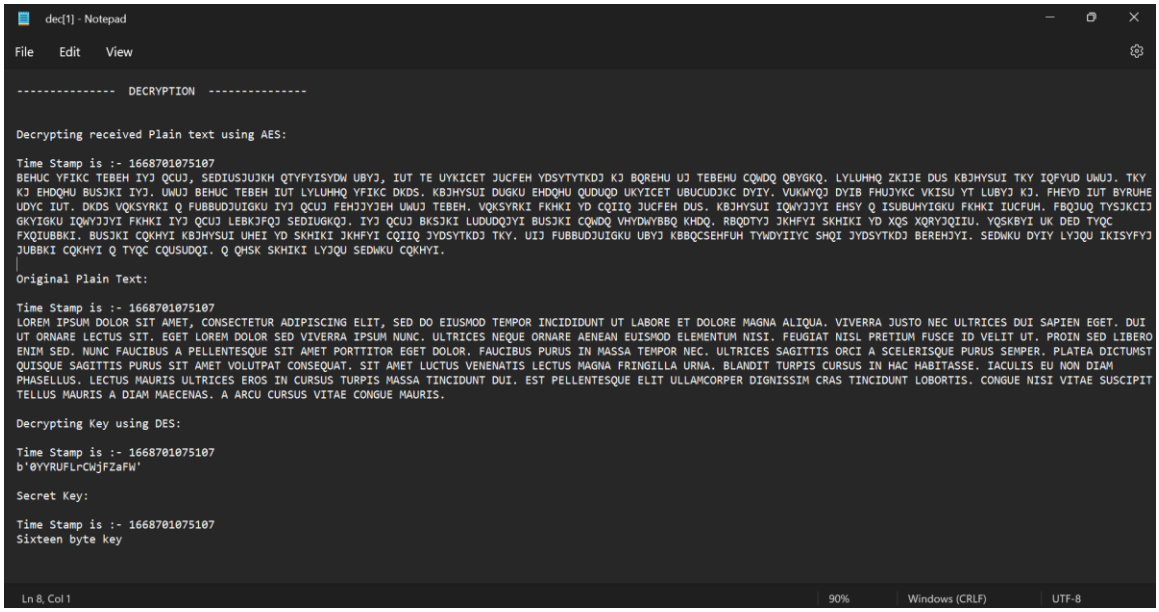

Figure 31 Decrypted text file

6. Conclusion

The proposed algorithm combines various modern and classical techniques such as Caesar cipher, vigenere cipher, DES and AES to create a more powerfull and complex algorithm which is comparably difficult to crack . Even in an unfortunate case of data leak such encryption agorithms can prevent breach of confidentiality , The algorithm can be improved by scrambling the key before ising the algorithm and integrating more of modern and classical encryption techniques. But we always have to keep in mind about the performance of the algorithm as we are going to be dealing with huge amounts of data.So it can be concluded that storing data in an unencrypted form is and can be potentially a threat to an organization it is therfore better to always to encrypt it and store it securely in a highly secured place.

7. Team Members' Contribution

Team Leader (Reg.No. & Name): FANINDRA NAYAK (20BIT0344)

Register Number	Name	Contribution / Role in this Project
20BIT0344	FANINDRA NAYAK	Coding
20BIT0025	NADELLA VENKATA GAGAN ROHITH	Coding and documentation
20BIT0347	DUDDU BALA GURU VENAKTA ARJUN	Algorithm design
20BIT0027	IMMANI SAI BHARGAV	Algorithm design and documentation
20BIT0063	TBS VINEETH	Literature survey and documentation

References

[1] J. Mahavir and A. Agrawal, "Implementation of hybrid cryptography algorithm," *International Journal Of Core Engineering & Management,* pp. 126-142, 2014.

[2] L. S and R. L, "Privacy preserving in data mining using hybrid approach," in *International Conference on Computational Intelligence and Communication Networks*, 2012.

[3] S. R. S, G. V and D. V, "A survey paper on data security in cloud computing," *International Journal of Computer Sciences and Engineering,* 2016.

[4] S. Parmeshwar, "A Survey Paper on Data Storage & Security in Cloud Computing," *Journal of the Gujarat Research Society,* 2019.

[5] J. Gaurav and S. Vikas, "Improving the security by using various cryptographic techniques in cloud computing," in *International Conference on Intelligent Computing and Control Systems*, 2017.

[6] F. C. Y. M. W. &. W. Q. Zhang, "Hybrid encryption algorithms for medical data storage security in cloud database," *International Journal of Database Management Systems,* 2019.

[7] K. S. N and V. A, "A survey on secure cloud: security and privacy in cloud computing," *American Journal of Systems and Software,* 2016.

[8] R. A and J. V, "Hybrid Cryptographic Based Approach for Privacy Preservation in Location-Based Services," in *International Conference on Wireless Intelligent and Distributed Environment for Communication,* 2018.

[9] G. R and K. R, "A Review Paper of Data Security in Cloud Computing," *Imperial Journal of Interdisciplinary Research,* 2016.

[10] C. M, V. D. M. A and K. P, "Secure cloud computing: Benefits, risks and controls," *Information Security for South Africa,* 2011.

[11] F. D. A, S. L. F, G. J. V, F. M. M and I. P. R, "Security issues in cloud environments: a survey," *International Journal of Information Security,* 2014.

[12] K. M. A, "A survey of security issues for cloud computing," *Journal of network and computer applications,* 2016.

[13] T. D, C. S, N. S and C. R. A, "Secure data sharing in the cloud," *Security, privacy and trust in cloud systems,* 2014.

[14] S. A and C. K, "Cloud security issues and challenges: A survey," *Journal of Network and Computer Applications*, 2017.

[15] K. S. S and T. R. R, "Security in cloud computing using cryptographic algorithms," *International Journal of Innovative Research in Computer and Communication Engineering*, 2015.