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

PROJECT REPORT

Topic	**Text-File Encryption using Hybrid AES-SCRYPT Algorithm**	
Team Members	18BIT0018	Devang Panwar
	18BIT0066	Deep Chaudhari
	18BIT0079	Sanjay B
Guided by	Dr. SHANTHARAJAH S P (Professor Grade 1, Department of Software and Systems Engineering)	
Course	Network and Information Security, ITE4001	

Text-File Encryption using Hybrid AES-SCRYPT Algorithm

Devang Panwar, Deep Chaudhari, Sanjay B

SITE School, VIT University, Vellore, Tamil Nadu

***Abstract*— The main encryption algorithms currently used are MD5, DES, RC4 and RSA, where RSA is the most widely used public key cryptography algorithm. Whereas many applications nowadays also use ECC in certain applications such as providing security in image processing applications and further. In this paper we are introducing a hybrid encryption method using AES and SCRYPT techniques where there will be two private keys to encrypt the file. We are using Scrypt in our proposal because Scrypt makes it more costly for breaking the password. Combining two powerful encryption techniques and a Private key system makes it really strong and costly to break the encryption.**

***Keywords*— AES, Scrypt, Private Key, Costly**

I. Introduction

The main encryption algorithms currently used are MD5, DES, RC4 and RSA, where RSA is the most widely used public key cryptography algorithm. RSA public key encryption algorithm was proposed by Rivest, Shamir and Adleman in 1977 (Schindler 2016). It is an encryption algorithm based on factorization of large integers. In the RSA encryption algorithm, both public and private keys can be used to encrypt data and can guarantee that the private key can not be derived from the public key (Azimi et al. 2017), and the plaintext can not be derived from the ciphertext (Jiang et al. 2017). The security of RSA depends on the factorization of large numbers (Khan et al. 2016). It is very difficult to factorize large numbers, while multiplying two large prime numbers is very simple to ensure the security of the RSA algorithm (Cui et al. 2013). However, in the process of encryption, the RSA algorithm needs a series of operations of large number multiplication, so the speed of operation has become a major defect of the RSA algorithm (Qian et al. 2017).

II. Motivation

We are trying to increase the cost required to break the AES encrypted files as attackers may use high efficient computers to break it.

Usage of Scrypt and Dual Private Key system not ensures safety but even if one private key is compromised it is harder to break the second one as both the keys are of 128 bits where it requires 2^{128} combinations to break one password.

III. Existing Methods

There are several block cipher methods available where blocks of data are processed each at a time. [2]O.Y.H.Cheung, K. H et.al they implemented IDEA(International Data Encryption Algorithm) where there idea was to take 64 bits plain text blocks and gives 64 bits cipher blocks using 128 bits key which resulted more powerful than DES algorithm, its primitive operations are of three distinct algebraic groups and multiplication modulo 2^16 + 1 provides independence between plaintext and ciphertext.[3]Nithin N, Anupkumar M et.al used FEAL(Fast encryption algorithm) to encrypt Grayscale images which is similar to DES.[4][5] also used Triple DES and DES techniques in their work where [4]they implemented these techniques on hardware for wireless local networks and [5] they used this technique to develop a Symmetric encryption algorithm. [8] Noura Aleisa compared both AES and Triple DES methods where the comparison is done based on factors like key size, block size, complexity,resources needed etc. The author found that the Triple DES algorithm is still beyond the capability of most attacks in the present day.But AES is undoubtedly better in terms of security and efficiency due to larger block size and key size.[10] Priyadarshini Patil et al also did this kind of comparison between DES, 3DES, AES, RSA and Blowfish on basis of Encryption time, Decryption time, Memory used, Avalanche effect and Entropy as parameters. In this study they found that the Memory required is smallest in blowfish whereas it is largest in RSA. If cryptographic strength is a major factor then AES is the best while If network bandwidth is a major factor then DES is a better choice. Also blowfish can be implemented if time and memory is a factor. Avalanche effect is highest in AES, RSA requires a large amount of memory and time for encryption and decryption. Also AES is not suitable for high network bandwidth transmission while DES & Blowfish lack security. [13]Henri Gilbert and Helena Handschuh did an analysis of SHA-256 and Sisters. The authors did this analysis against collision attacks. They attacked SHA 256, SHA384,SHA 512 with Chaubaud, Joux, Dobbertin linear attacks. They also performed differential attacks on the triad. Chabaud and Joux's attack, nor Dobbertin-style attacks. Differential and linear attacks don't apply on the underlying structure of the SHA family. The number of rounds to state register length ratio, which represents the number of "full rotations" of the state register during each compression function computation, is much lower for SHA 256,SHA 384 and SHA 512. They found that the complexity of a collision search is very low(just 2 to the power 64 iterations). Block chaining also made its mark as a secure system [6]. Daniar Heri Kurniawan et al discussed about implementation of Double Chaining Algorithm (DCA) for faster and safer encryption and decryption in which they used symmetric algorithm which applies block chaining with 16 byte block length by using 128 - 256 bits key size to encrypt the data. They found that their method of implementing block chain technique showed a comparatively great avalanche effect compared to Triple DES algorithm. This shows that block chain is also a great alternative for safer and secure encryption techniques.[7] AES is also used for purposes of secure cloud storage, Babitha M.P. et al used Advanced Encryption System(AES) in their work for secure cloud storage. They did this by Addressing different data security and privacy protection issues in a cloud computing environment and implementing AES-128 is used to increase data security and confidentiality. They used Advanced Encryption Standard (AES) with key length 128 bits and block size 128 bits is used to encrypt the data that is to be loaded to the cloud. The main disadvantage of using this encryption for this purpose is that the delay in uploading becomes drastic if the size of the file is large. There are also many Stream ciphers like RSA which are majorly used in the industry which are even more secure for particular purposes.[9]Suli Wang et al also used RSA in their work. RSA is an asymmetric algorithm which uses a public and private key pair to encrypt and decrypt the file where the key size is more than 1024 bits and minimum block length is 512. RSA algorithm is suitable for encryption of small files and it is tougher to decipher the data since it has a large key and block size. The main disadvantage is ,RSA takes more time in encryption and decryption when compared to DES due to which it is not suitable for hardware as well as software implementation.[15][16][18] SCRYPT is also widely used in the industry with combination of other encrypting techniques in order to increase the cost of attacks on the system. [11][13][14] SHA family and MD5 are also widely used hashing algorithms. But the attackers made their way through such strong hashing algorithms already.[11] showa one of the ways in which Xiaoyun Wang et al broke MD5 and some other hash functions such as SHA1 and so on. They did this by finding collisions within two iterations using a new powerful attack on MD5. Unlike all other differential attacks this method is not using XOR operation as their differential method to attack rather, they are using modular integer subtraction as their measure of attack. MD5 is vulnerable to many differential cryptanalysis which make it unsuitable for hashing.In [12] Friedrich et al introduced High Speed implementation of bcrypt password search using special purpose hardware. This is proven as a novel flexible high speed implementation of a bcrypt password search system on a low-power Xilinx Zynq 7020 FPGA. They used multiple clock domains in order to reduce resource usage.They used 40 parallel bcrypt cores on FPGA’s to result in efficient clock speeds. This algorithm showed 127% power efficiency running on low power and resources are compactly used throughout the process.The major problem is that hash functions are faster to evaluate so they enable faster attacks in password attacks.[15] Percival et al proposed a scrypt password based key derivation function.The scrypt function aims to decrease the advantage of attackers computational power to decrease the cost of brute force attack. The main objective of this work is to serve as a stable reference for the documents making use of scrypt. They used C language to implement the salsa20/8 program and other scrypt mix algorithms. The remarks on this process is, By nature and depending on parameters, running the scrypt algorithm may require bigger amounts of memory. Systems should protect against a denial-of-service attack resulting from attackers presenting unreasonably large parameters.[16] Anne Barshun et al deployed an attack on scrypt using cache timing attack. They used the vulnerability in the scrypt password hashing of dependency on the original key to do this attack. They exploited the inter-process leakage through memory cache as their primitive to do cache time attack. They found that they can exploit the small property of temporary memory storage of cache as their parameter and capture the memory leaks occurred during computation. Scrypt uses the password to build a large array of hashes and access them with an index derived from the original key which makes it dependent on the original key. These cache attacks are not only limited to scrypt, [17] Eran Tromer et al deployed this attack on AES and even suggested some countermeasures for that. They used the property of cache storing the frequent data and exploited this feature to extract the plaintext itself and sometimes key. Proposed several countermeasures for this attack such as avoiding memory access, alternative lookup tables, Data independent memory access pattern, Application specific algorithm masking...etc. In all of them avoiding memory access gave good reach as a countermeasure. The major disadvantage is that this attack only works if there are page faults otherwise this attack is just empty. Efficient computing these days shows very less page faults as possible.[18] Joel et al showed practically that scrypt is memory hard.They attacked the scrypt with brute force, and gave the first non-trivial unconditional lower bound on ccmem for scrypt in the parallel random oracle model, and their bound already achieves optimal ccmem of $\Omega(w \cdot n2)$. Where ccmen is similar to cumulative pebbling complexity. It is true that scrypt is memory hard but if the attacker has access to special purpose hardware like ASCI which exploits the use of parallelism, pipelining, and amortization can make the computation a bit easy.

IV. PROPOSED METHOD

A. System Architecture

In symmetric cryptography, there is a single key (called secret key or private key) that is used to encrypt as well as decrypt the data. The parties that need to communicate with each other must have the same secret key. The system architecture is shown in Fig 1. The proposed system is performing in the following procedures: Fig1 shows the encryption and decryption process of plaintext file. Encryption takes place at sender side while decryption at receiver side. The input of the encryption process is a plaintext file and that of the decryption process is the cipher text file. First plain text file is passed through the AES encryption algorithm which encrypts the plain text file using a “specially generated” key which is hashed using Scrypt hashing function and then produces a ciphertext file i.e. encrypted file is transmitted. At the end of decryption the input cipher text file is passed through the AES decryption algorithm which can decrypt the cipher text file i.e. encrypted file using the same key as that of encryption finally we get the original plain text file.

B. Algorithms used for Proposed work

i] Special key generation:

- Two separate keys are generated by data admin as shown in Fig 1
- These two keys are of 128 bits which gives 16 character long passwords.
- Now we combine these keys to generate a master key as shown in Fig 2 which in turn we use for encryption and decryption.
- According to the diagram we generate the master key.
- Key expansion is done as follows. We append every 4th bit in the order to the resulting 64 bit key to expand it to 128 bit key.

ii] Encryption and Decryption(AES-128):

- Infer the set of round keys from the special generated key.
- Initialize the state array with the block data (plaintext).
- Add the initial round key to the starting state array.
- Each round has following steps:

  -Add round key
  -Substitute bytes
  -Shift rows
  -Mix columns
  -Add round key
- Perform nine rounds of state manipulation as per Fig 3.
- Perform the tenth and final round of state manipulation.
- Copy the final state array out as the encrypted data (ciphertext).

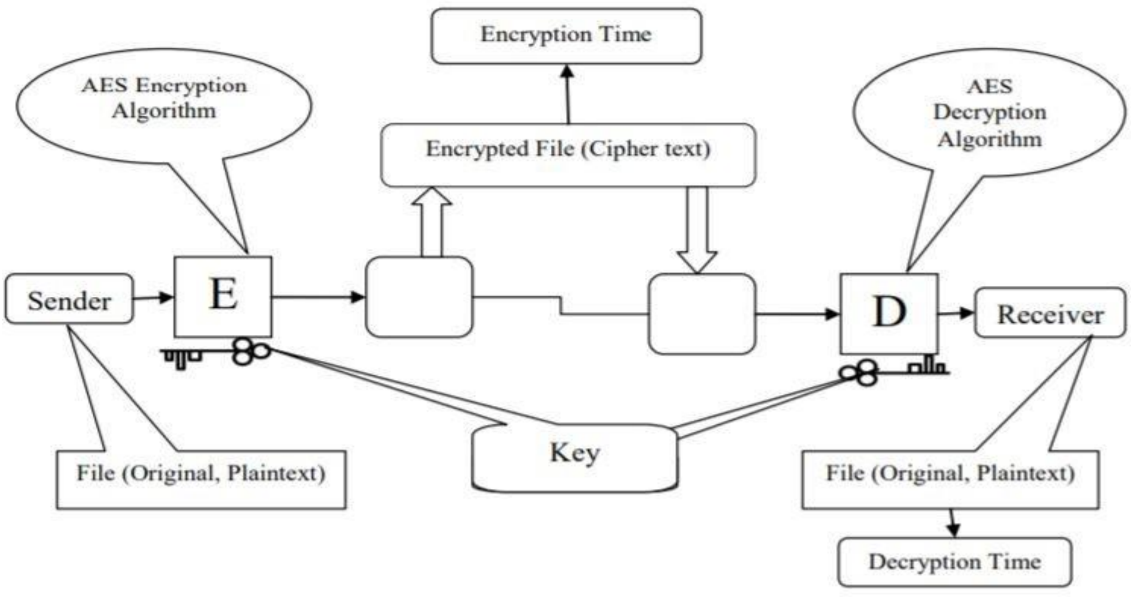

Encryption Time
AES Encryption Algorithm
Encrypted File (Cipher text)
AES Decryption Algorithm
Sender
E
D
Receiver
Key
File (Original, Plaintext)
File (Original, Plaintext)
Decryption Time

Fig 1

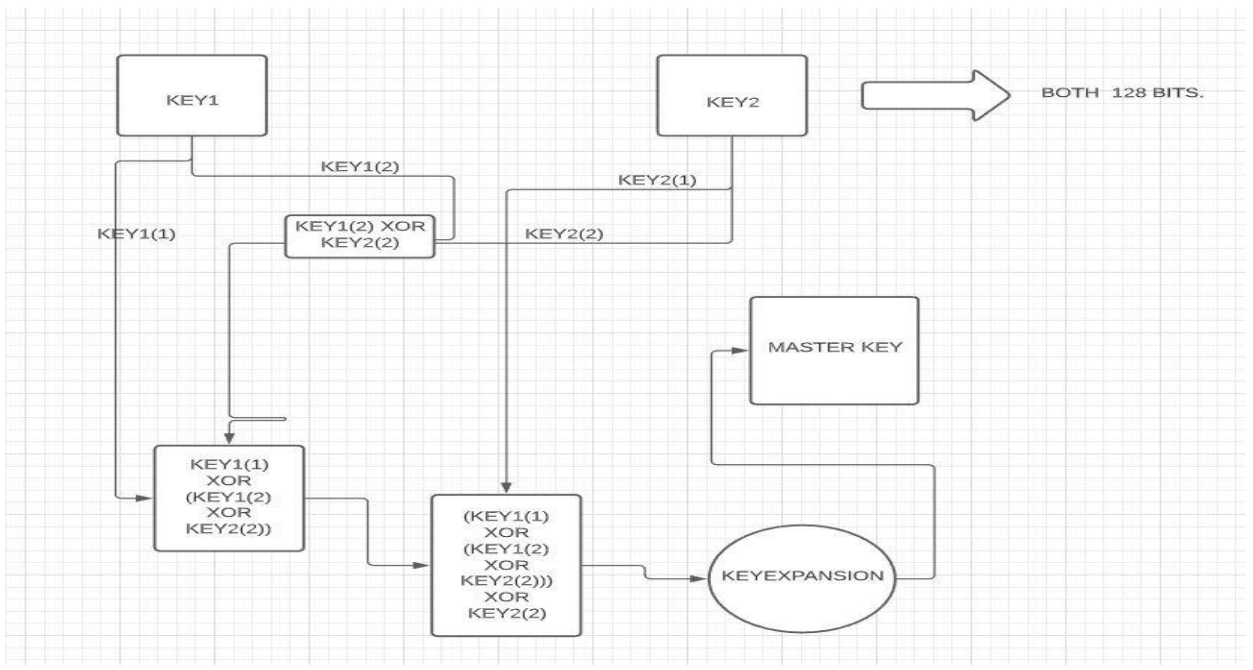

KEY1
KEY2
BOTH 128 BITS.
KEY1(2)
KEY2(1)
KEY1(1)
KEY1(2) XOR KEY2(2)
KEY2(2)
MASTER KEY
KEY1(1) XOR (KEY1(2) XOR KEY2(2))
(KEY1(1) XOR (KEY1(2) XOR KEY2(2))) XOR KEY2(2)
KEYEXPANSION

Fig 2

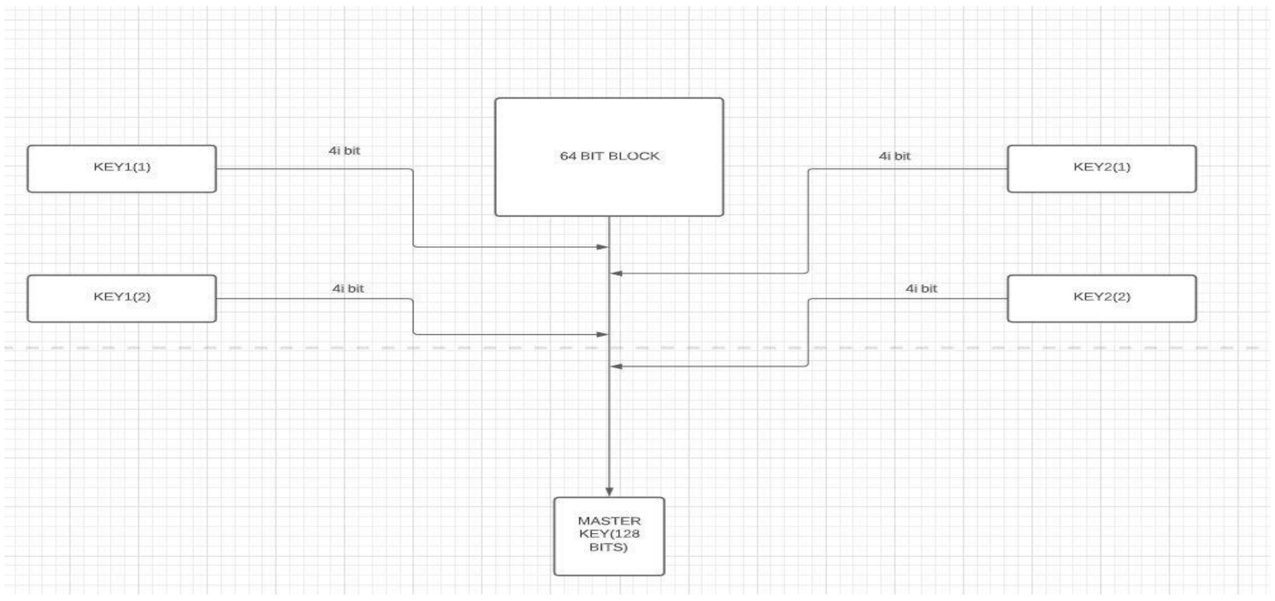

64 BIT BLOCK
KEY1(1)
4i bit
KEY2(1)
4i bit
KEY1(2)
4i bit
KEY2(2)
4i bit
MASTER KEY(128 BITS)

Fig 3

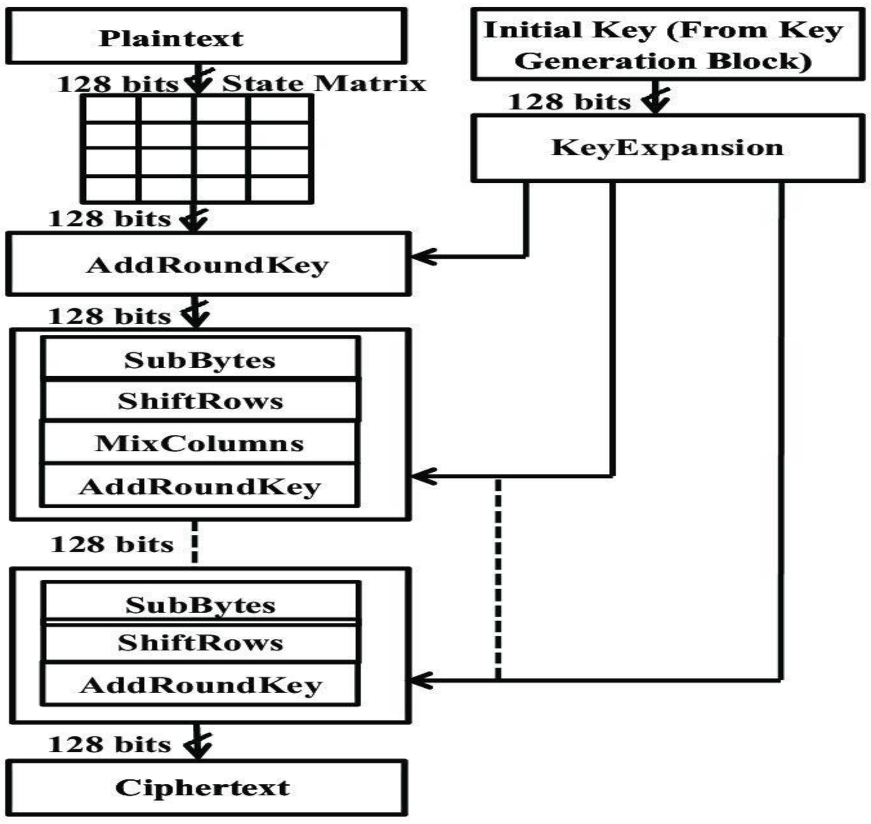

Plaintext
128 bits
State Matrix
Initial Key (From Key Generation Block)
128 bits
KeyExpansion
128 bits
AddRoundKey
128 bits
SubBytes
ShiftRows
MixColumns
AddRoundKey
128 bits
SubBytes
ShiftRows
AddRoundKey
128 bits
Ciphertext

Fig 4

V. Comparative Analysis

I] DES

Data Encryption Standard (DES) is a symmetric key block cipher. The key length is 56 bits and block size is 64 bit length. it's susceptible to key attack when a weak keys used. DES was found in 1972 by IBM using the data encryption algorithm. it had been adopted by the govt of the USA as a standard encryption algorithm. It began with a 64 bit key then the NSA put a restriction to use of DES with a 56- bit key length, hence DES discards 8 bits of the 64 bit key then uses the compressed 56 bit key derived from 64 bit key to encrypt data in block size of 64- bits .DES can operate in several modes - CBC, ECB, CFB and OFB, making it flexible. In 1998 the supercomputer DES cracker, with the assistance of lakh's of distributed PCs on the web , cracked DES in 22h.

II] Triple DES

In cryptography, Triple DES is additionally called Triple data encryption Algorithm which is a block cipher. Triple data encryption Standard (3DES) was first published in 1998 which gets its name so because it applies DES cipher 3 times to every block of data, Encryption – Decryption – Encryption using DES. The key length is 112 bits or 168 bits and block size is 64 bit length. As a result of the increasing computational power available currently and weak of the initial DES cipher, it absolutely was subject to brute force attacks and various cryptanalytic attacks; Triple DES was designed to produce a comparatively simple method of skyrocketing the key size of DES to shield against such attacks, without designing a very new block cipher algorithm

III] AES

Advance Encryption Standard (AES) algorithm was developed in 1998 by Joan Daemen and Vincent Rijmen, which may be a symmetric key block cipher.AES algorithm, supports any combination of data and key length of 128, 192, and 256 bits. AES allows a 128 bit data length which will be split into four basic operational blocks. These blocks are arranged as array of bytes and organized as a matrix of the order of 4×4 which is additionally called as state and subject to rounds where various transformations are done. For full encryption, the amount of rounds used is variable N = 10, 12, 14 for key length of 128,192 and 256 respectively. Each round of AES uses a permutation and substitution network, and is suitable for both hardware and software implementation.

IV] Blowfish

Blowfish was first published in 1993 .It is a symmetric key block cipher with key length variables from 32 to 448 bits and block size of 64 bits. Its structure is a feistel network. Blowfish is a symmetric block cipher which will be used as an off-the-cuff replacement for DES or IDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and commercial use. Blowfish was designed by Bruce Schneier as a quick , free alternative to existing encryption algorithms. From then it's been analyzed considerably, and it's slowly gaining popularity as a strong encryption algorithm.

Blowfish isn't patented, has free license and is freely available for all uses.

V] RSA

RSA was founded in 1977 as a public key cryptosystem. RSA is an asymmetric cryptographic algorithm named after its founders Rivest, Shamir &Adelman. It generates two keys: public key for encryption and private key to decrypt message .RSA algorithm contains three steps, the first step is key generation which is to be used as key to encrypt and decrypt data, step two is encryption, where actual process of conversion of plaintext to cipher text is being administered and third step is decryption, where encrypted text is converted in to plain text at other side.RSA is predicated on factoring problem of finding product of two large prime numbers. Key size is 1024 to 4096 bits.

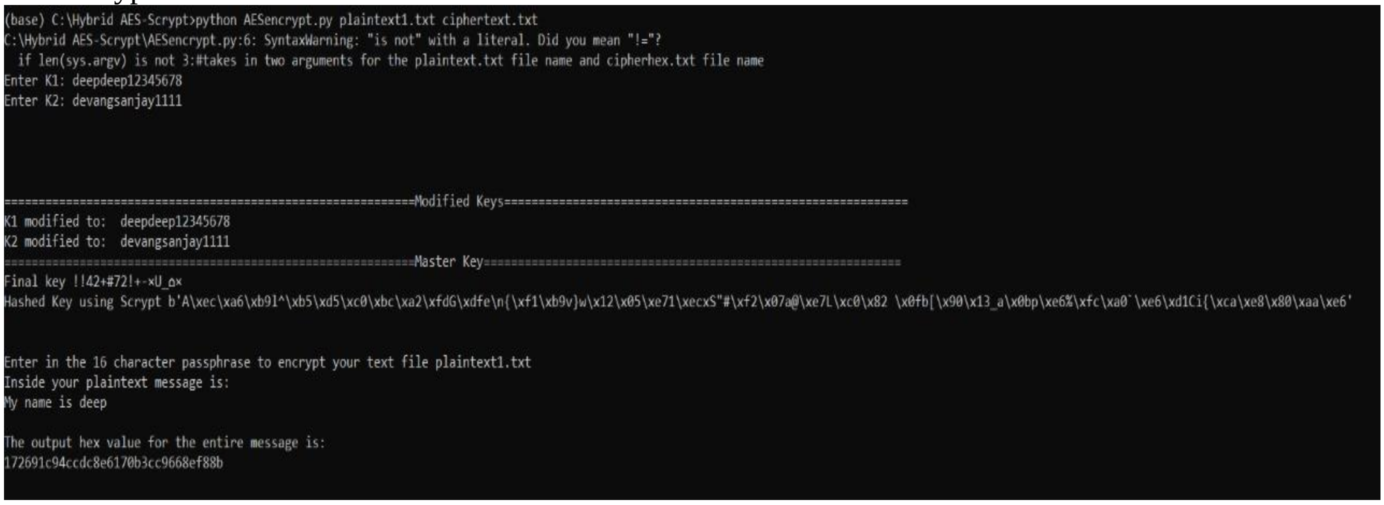
VI. IMPLEMENTATION

The implementation is almost very similar to AES. A machine with updated OS and good memory is enough for the implementation of our program. Minimum Software details required for the implementation of the our AES-Scrypt encryption and decryption are as follows:

1) Core i3
2) Anaconda3 environment
3) 4 GB RAM
4) Python 3.2 and above
5) Windows 7/ Mac OS 10

Screenshots:

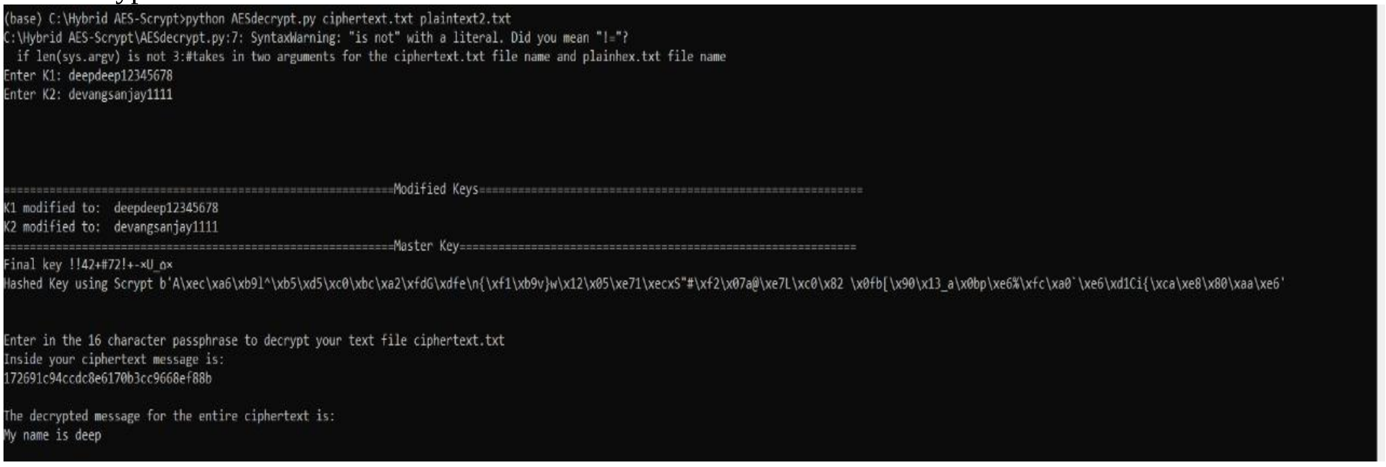
Encryption-



```
(base) C:\Hybrid AES-Scrypt>python AESencrypt.py plaintext1.txt ciphertext.txt
C:\Hybrid AES-Scrypt\AESencrypt.py:6: SyntaxWarning: "is not" with a literal. Did you mean "!="?
  if len(sys.argv) is not 3:#takes in two arguments for the plaintext.txt file name and cipherhex.txt file name
Enter K1: deepdeep12345678
Enter K2: devangsanjay1111




============================================================Modified Keys============================================================
K1 modified to:  deepdeep12345678
K2 modified to:  devangsanjay1111
============================================================Master Key============================================================
Final key !!42+#72!+-×U_ó×
Hashed Key using Scrypt b'A\xec\xa6\xb91^\xb5\xd5\xc0\xbc\xa2\xfdG\xdfe\n{\xf1\xb9v}w\x12\x05\xe71\xecxS"#\xf2\x07a@\xe7L\xc0\x82 \x0fb[\x90\x13_a\x0bp\xe6%\xfc\xa0`\xe6\xd1Ci{\xca\xe8\x80\xaa\xe6'


Enter in the 16 character passphrase to encrypt your text file plaintext1.txt
Inside your plaintext message is:
My name is deep

The output hex value for the entire message is:
172691c94ccdc8e6170b3cc9668ef88b
```

Fig: 5

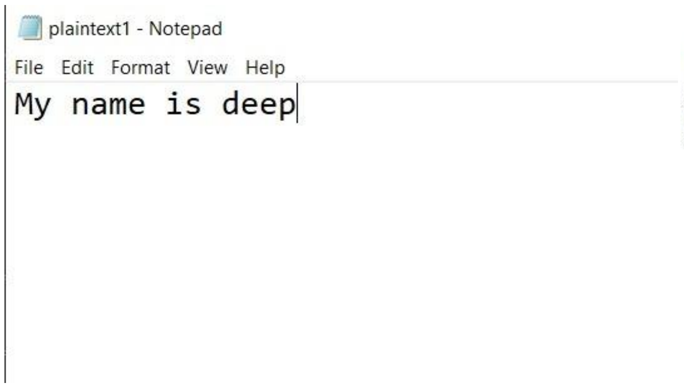
Decryption-



```
(base) C:\Hybrid AES-Scrypt>python AESdecrypt.py ciphertext.txt plaintext2.txt
C:\Hybrid AES-Scrypt\AESdecrypt.py:7: SyntaxWarning: "is not" with a literal. Did you mean "!="?
  if len(sys.argv) is not 3:#takes in two arguments for the ciphertext.txt file name and plainhex.txt file name
Enter K1: deepdeep12345678
Enter K2: devangsanjay1111




============================================================Modified Keys============================================================
K1 modified to:  deepdeep12345678
K2 modified to:  devangsanjay1111
============================================================Master Key============================================================
Final key !!42+#72!+-×U_ó×
Hashed Key using Scrypt b'A\xec\xa6\xb91^\xb5\xd5\xc0\xbc\xa2\xfdG\xdfe\n{\xf1\xb9v}w\x12\x05\xe71\xecxS"#\xf2\x07a@\xe7L\xc0\x82 \x0fb[\x90\x13_a\x0bp\xe6%\xfc\xa0`\xe6\xd1Ci{\xca\xe8\x80\xaa\xe6'


Enter in the 16 character passphrase to decrypt your text file ciphertext.txt
Inside your ciphertext message is:
172691c94ccdc8e6170b3cc9668ef88b

The decrypted message for the entire ciphertext is:
My name is deep
```

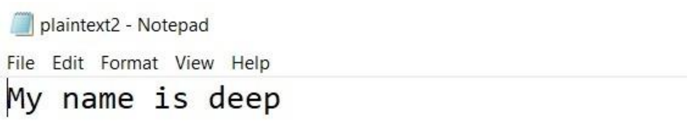
Fig: 6

Plaintext1.txt



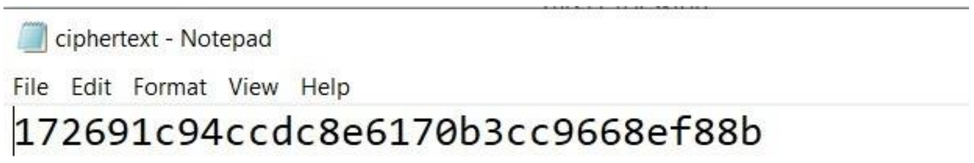
My name is deep

Plaintext2.txt



My name is deep

Ciphertext.txt



```
172691c94ccdc8e6170b3cc9668ef88b
```

Sample Code:

AESencrypt.py

```
from AESencryptfunc import * #import
AESencryptfunc module to use functions
created for this program
import math #import math module to use
function such as ceiling
import scrypt

#check that script is running with the two
text files as the two parameters or else
quit
if len(sys.argv) is not 3:#takes in two
arguments for the plaintext.txt file name
and cipherhex.txt file name
sys.exit("Error, script needs two
command-line arguments. (Plaintext.txt
File and cipherhex.txt File)")

# set password to be a 16 characters, 16
characters * 8 bits = 128 bits strength
PassPhrase=""
def string2bits(par=""):
    return [bin(ord(x))[2:].zfill(8) for x
in par]
def xor_two_str(a,b):
    return ''.join([hex(ord(a[i%len(a)]) ^
ord(b[i%(len(b))]))[2:] for i in
range(max(len(a), len(b)))])

def Master_Key_128(K1,K2):
    #Key Mixing-Start
    K1_1=string2bits(K1)[0:8]
    K1_2=string2bits(K2)[8:16]
    K2_1=string2bits(K2)[0:8]
    K2_2=string2bits(K2)[8:16]
    result=[]
    for i in range(8):

result.append(xor_two_str(K2_1[i],xor_two_
str(K1_1[i],xor_two_str(K1_2[i],K2_2[i])))
)
    #Key  Mixing-End
    #Key Expansion-Start
    K1_1_result=''.join(i for i in K1_1)
    K1_2_result=''.join(i for i in K1_2)
    K2_1_result=''.join(i for i in K2_1)
    K2_2_result=''.join(i for i in K2_2)
    temp=""

    for i in range(1,17):

temp=temp+"".join(K1_1_result[4*i-1])+"".j
oin(K1_2_result[4*i-1])+"".join(K2_1_resul
t[4*i-1])+"".join(K2_2_result[4*i-1])

    for i in range(8):
        result.append(("".join(j for j in
list(temp)[8*i:8*i+8])))
    #KeyExpansion-End
    #Converting Final Key into characters
from binary
    finalkey=''
    for i in range(16):
        if(int(result[i],2)<32):

finalkey=finalkey+chr(int(result[i],2)+33)
        else:

finalkey=finalkey+chr(int(result[i],2))

    salt = 'd7d42612c5ff4625'
    hashed_key=scrypt.hash(finalkey,salt)
    return [finalkey,hashed_key]


K1=input("Enter K1: ")
K2=input("Enter K2: ")
print("\n")
k=Master_Key_128(K1,K2)
print("Hashed Key using Scrypt",k[1])
print("\n")
while(len(PassPhrase)!=16):
    print("Enter in the 16 character
passphrase to encrypt your text file %s"
%sys.argv[1])
    PassPhrase=k[0] #takes in user input
of char, eg. "Iwanttolearnkung"
    if(len(PassPhrase)<16):#check if less
than 16 characters, if so add one space
character until 16 chars
        while(len(PassPhrase)!=16):
            PassPhrase=PassPhrase+"\00"
    if(len(PassPhrase)>16):#check if
bigger than 16 characters, if so then
truncate it to be only 16 chars from
[0:16]
        print("Your passphrase was larger
than 16, truncating passphrase.")
        PassPhrase=PassPhrase[0:16]

#open plaintext.txt file to read and
encrypt
file=open(sys.argv[1], "r")
message=(file.read())
print("Inside your plaintext message
is:\n%s\n" % message)
file.close()

message=BitVector(textstring=message)
message=message.get_bitvector_in_hex()
replacementptr=0
while(replacementptr<len(message)):

if(message[replacementptr:replacementptr+2
]=='0a'):

message=message[0:replacementptr]+'0d'+mes
sage[replacementptr:len(message)]
        replacementptr=replacementptr+4
    else:
        replacementptr=replacementptr+2

message=BitVector(hexstring=message)
message=message.get_bitvector_in_ascii()
#set up some parameters
start=0#set starting pointer for the part
to encrypt of the plaintext
end=0#set ending pointer for the part to
encrypt of the plaintex
length=len(message)#check the entire size
of the message
loopmsg=0.00#create a decimal value
loopmsg=math.ceil(length/16)+1#use formula
to figure how long the message is and how
many 16 character segmentss must be
encrypted
outputhex=""#setup output message in hex

#need to setup roundkeys here
PassPhrase=BitVector(textstring=PassPhrase
)
roundkey1=findroundkey(PassPhrase.get_bitv
ector_in_hex(),1)
roundkey2=findroundkey(roundkey1,2)
roundkey3=findroundkey(roundkey2,3)
```

```
roundkey4=findroundkey(roundkey3,4)
roundkey5=findroundkey(roundkey4,5)
roundkey6=findroundkey(roundkey5,6)
roundkey7=findroundkey(roundkey6,7)
roundkey8=findroundkey(roundkey7,8)
roundkey9=findroundkey(roundkey8,9)
roundkey10=findroundkey(roundkey9,10)
roundkeys=[roundkey1,roundkey2,roundkey3,r
oundkey4,roundkey5,roundkey6,roundkey7,rou
ndkey8,roundkey9,roundkey10]

#set up FILEOUT to write
FILEOUT = open(sys.argv[2], 'w')

# set up the segement message loop
parameters
for y in range(1, loopmsg): # loop to
encrypt all segments of the message
    if(end+16<length): #if the end pointer
is less than the size of the message, then
set the segment to be 16 characters
        plaintextseg = message[start:end +
16]
    else: #or else if the end pointer is
equal to or greator than the size of the
message
        plaintextseg =
message[start:length]
        for z in
range(0,((end+16)-length),1): #run a while
loop to pad the message segement to become
16 characters, if it is 16 already the
loop will not run
            plaintextseg =
plaintextseg+"\00"

#plaintextseg2=BitVector(textstring=plaint
extseg)

#print(plaintextseg2.get_bitvector_in_hex(
))

    #add round key zero/ find round key
one
    bv1 =
BitVector(textstring=plaintextseg)
    bv2 = PassPhrase
    resultbv=bv1^bv2
    myhexstring =
resultbv.get_bitvector_in_hex()

    for x in range(1, 10):  # loop through
9 rounds
        # sub byte
        myhexstring =
resultbv.get_bitvector_in_hex()
        temp1=subbyte(myhexstring)

        # shift rows
        temp2=shiftrow(temp1)

        # mix column
        bv3 = BitVector(hexstring=temp2)
        newbvashex=mixcolumn(bv3)

newbv=BitVector(hexstring=newbvashex)

        #add roundkey for current round
        bv1 = BitVector(bitlist=newbv)
        bv2 =
BitVector(hexstring=roundkeys[x-1])
        resultbv = bv1 ^ bv2
        myhexresult =
resultbv.get_bitvector_in_hex()

    #start round 10
    # sub byte round 10
    myhexstring =
resultbv.get_bitvector_in_hex()
    temp1=subbyte(myhexstring)

    # shift rows round 10
    temp2=shiftrow(temp1)

    # add round key round 10
    newbv = BitVector(hexstring=temp2)
    bv1 = BitVector(bitlist=newbv)
    bv2 =
BitVector(hexstring=roundkeys[9])
    resultbv = bv1 ^ bv2
    myhexstring =
resultbv.get_bitvector_in_hex()

    #set encrypted hex segement of message
to output string
    outputhextemp =
resultbv.get_hex_string_from_bitvector()
    FILEOUT.write(outputhextemp)
    start = start + 16 #increment start
pointer
    end = end + 16 #increment end pointer

# encrypted output hex string to specified
cipherhex file
FILEOUT.close()

file2=open(sys.argv[2], "r")
print("The output hex value for the entire
message is:\n%s\n" % file2.read())
file2.close()
```

AESdecrypt.py

```
from AESdecryptfunc import * #import
AESdecryptfunc module to use functions
created for this program
import math #import math module to use
function such as ceiling
import io
import scrypt

#check that script is running with the two
text files as the two parameters or else
quit
if len(sys.argv) is not 3:#takes in two
arguments for the ciphertext.txt file name
and plainhex.txt file name
    sys.exit("Error, script needs two
command-line arguments. (Ciphertext.txt
File and plainhex.txt File)")

PassPhrase=""

def string2bits(par=""):
    return [bin(ord(x))[2:].zfill(8) for x
in par]
def xor_two_str(a,b):
    return ''.join([hex(ord(a[i%len(a)]) ^
ord(b[i%(len(b))]))[2:] for i in
range(max(len(a), len(b)))])

def Master_Key_128(K1,K2):
    #Key Mixing-Start
    K1_1=string2bits(K1)[0:8]
    K1_2=string2bits(K2)[8:16]
    K2_1=string2bits(K2)[0:8]
    K2_2=string2bits(K2)[8:16]
    result=[]
    for i in range(8):

result.append(xor_two_str(K2_1[i],xor_two_
str(K1_1[i],xor_two_str(K1_2[i],K2_2[i])))
)
    #Key  Mixing-End
    #Key Expansion-Start
    K1_1_result=''.join(i for i in K1_1)
    K1_2_result=''.join(i for i in K1_2)
    K2_1_result=''.join(i for i in K2_1)
    K2_2_result=''.join(i for i in K2_2)
    temp=""

    for i in range(1,17):

temp=temp+"".join(K1_1_result[4*i-1])+"".j
oin(K1_2_result[4*i-1])+"".join(K2_1_resul
t[4*i-1])+"".join(K2_2_result[4*i-1])

    for i in range(8):
        result.append(("".join(j for j in
list(temp)[8*i:8*i+8])))
    #KeyExpansion-End
    #Converting Final Key into characters
from binary
    finalkey=''
    for i in range(16):
        if(int(result[i],2)<32):

finalkey=finalkey+chr(int(result[i],2)+33)
        else:

finalkey=finalkey+chr(int(result[i],2))

    salt = 'd7d42612c5ff4625'
    hashed_key=scrypt.hash(finalkey,salt)
    return [finalkey,hashed_key]


K1=input("Enter K1: ")
K2=input("Enter K2: ")
print("\n")
k=Master_Key_128(K1,K2)
print("Hashed Key using Scrypt",k[1])
print("\n")

while(len(PassPhrase)!=16):
    print("Enter in the 16 character
passphrase to decrypt your text file %s"
%sys.argv[1])
    PassPhrase=k[0] #takes in user input
of char, eg. "Iwanttolearnkung"
    if(len(PassPhrase)<16):#check if less
than 16 characters, if so add one space
character until 16 chars
        while(len(PassPhrase)!=16):
            PassPhrase=PassPhrase+"\00"
    if(len(PassPhrase)>16):#check if
bigger than 16 characters, if so then
truncate it to be only 16 chars from
[0:16]
        print("Your passphrase was larger
than 16, truncating passphrase.")
        PassPhrase=PassPhrase[0:16]

#open ciphertext.txt file to read and
decrypt
file=open(sys.argv[1], "r")
message=(file.read())
print("Inside your ciphertext message
is:\n%s\n" % message)
file.close()

#set up some parameters
start=0#set starting pointer for the part
to decrypt of the ciphertext
end=32#set ending pointer for the part to
decrypt of the plaintex
length=len(message)#check the entire size
of the message
loopmsg=0.00#create a decimal value
```

```
loopmsg=math.ceil(length/32)+1#use formula
to figure how long the message is and how
many 16 character segmentss must be
decrypted
outputhex=""#setup output message segment
in hex
asciioutput=""#setup compilation of output
message in ascii

#need to setup roundkeys here
PassPhrase=BitVector(textstring=PassPhrase
)
roundkey1=findroundkey(PassPhrase.get_bitv
ector_in_hex(),1)
roundkey2=findroundkey(roundkey1,2)
roundkey3=findroundkey(roundkey2,3)
roundkey4=findroundkey(roundkey3,4)
roundkey5=findroundkey(roundkey4,5)
roundkey6=findroundkey(roundkey5,6)
roundkey7=findroundkey(roundkey6,7)
roundkey8=findroundkey(roundkey7,8)
roundkey9=findroundkey(roundkey8,9)
roundkey10=findroundkey(roundkey9,10)
roundkeys=[roundkey1,roundkey2,roundkey3,r
oundkey4,roundkey5,roundkey6,roundkey7,rou
ndkey8,roundkey9,roundkey10]

FILEOUT = io.open(sys.argv[2], 'w',
encoding='utf-8')

# set up the segement message loop
parameters
for y in range(1, loopmsg): # loop to
encrypt all segments of the message
    plaintextseg = message[start:end]

    # add round key
    bv1 =
BitVector(hexstring=plaintextseg)
    bv2 =
BitVector(hexstring=roundkeys[9])
    resultbv = bv1 ^ bv2
    myhexstring =
resultbv.get_bitvector_in_hex()

    #inverse shift row
    myhexstring=invshiftrow(myhexstring)

    #inverse subbyte
    myhexstring=invsubbyte(myhexstring)

    for x in range(8, -1, -1):
        # add roundkey for current round
        bv1 =
BitVector(hexstring=myhexstring)
        bv2 =
BitVector(hexstring=roundkeys[x])
        resultbv = bv1 ^ bv2
        myhexstring =
resultbv.get_bitvector_in_hex()

        # mix column
        bv3 =
BitVector(hexstring=myhexstring)
        myhexstring=invmixcolumn(bv3)

        # shift rows
        myhexstring =
invshiftrow(myhexstring)

        # sub byte
        myhexstring =
invsubbyte(myhexstring)

    #add initial round key
    bv1 = BitVector(hexstring=myhexstring)
    bv2 = PassPhrase
    resultbv = bv1 ^ bv2
    myhexstring =
resultbv.get_bitvector_in_hex()

    start = start + 32 #increment start
pointer
    end = end + 32 #increment end pointer

    replacementptr = 0
    while (replacementptr <
len(myhexstring)):
        if
(myhexstring[replacementptr:replacementptr
+ 2] == '0d'):
            myhexstring =
myhexstring[0:replacementptr] +
myhexstring[replacementptr+2:len(myhexstri
ng)]
        else:
            replacementptr =
replacementptr + 2

    outputhex =
BitVector(hexstring=myhexstring)
    asciioutput =
outputhex.get_bitvector_in_ascii()

asciioutput=asciioutput.replace('\x00','')
    FILEOUT.write(asciioutput)

FILEOUT.close()

file2=io.open(sys.argv[2], "r",
encoding='utf-8')
print("The decrypted message for the
entire ciphertext is:\n%s\n" %
file2.read())
file2.close()
```

VII. EVALUATION PARAMETERS

Encryption Time

The time taken to convert plaintext to ciphertext is encryption time. Encryption time depends upon key size, plaintext block size and mode. In our experiment we have measured encryption time in milliseconds. Encryption time impacts performance of the system. Encryption time must be less, making the system fast and responsive.

Decryption Time

The time to recover plaintext from ciphertext is called decryption time. The decryption time is desired to be less similar to encryption time to make the system responsive and fast. Decryption time impacts the performance of a system. In our experiment, we have measured decryption time is milliseconds.

Memory Usage

Different encryption techniques require different memory sizes for implementation. This memory requirement depends on the number of operations to be done by the algorithm, key size used, initialization vectors used and type of operations. The memory used impacts the cost of the system. It is desirable that the memory required should be as small as possible.

Complexity

Complexity of an encryption algorithm is confusion and diffusion of the code.In cryptography, **confusion** and **diffusion** are two properties of the operation of a secure cipher.

Confusion

Confusion means that cipher text should totally depend on several parts of the key but not restraining to only certain parts of the key. The main property of confusion masks or hides the relationship between key and cipher text.

This property makes it difficult to find the key from the ciphertext and if a single bit in a key is changed, the calculation of the values of most or all of the bits in the ciphertext will be affected.

Confusion increases the ambiguity of ciphertext and it is used by both block and stream ciphers.

Diffusion

Diffusion means that if we change a single bit of the plaintext, then (statistically) half of the bits in the ciphertext should change, and similarly, if we change one bit of the ciphertext, then approximately one half of the plaintext bits should change.Since a bit can have only two states, when they are all re-evaluated and changed from one seemingly random position to another, half of the bits will have changed state.

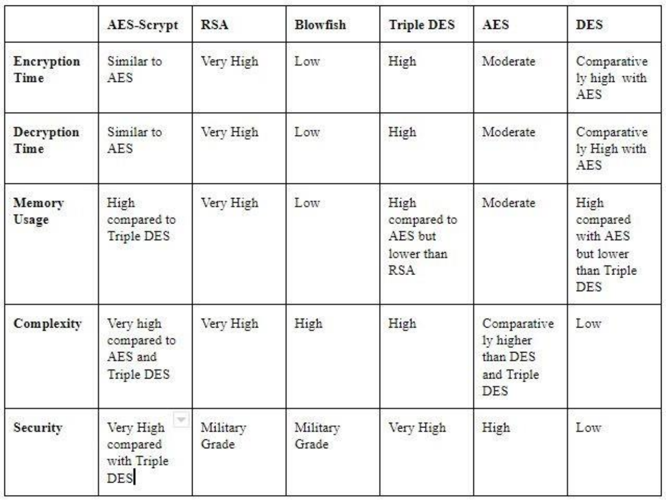
The idea of diffusion is to hide the relationship between the ciphertext and the plain text.

This will make it hard for an attacker who tries to find out the plain text and it increases the redundancy of plain text by spreading it across the rows and columns; it is achieved through transposition of algorithms and it is used by block ciphers only.

VIII. Comparison

As Shown in Table.1 AES-Scrypt hybrid algorithm is similar to AES algorithm but a bit slow, which is negligible than AES as it is using two keys to process which are of 128 bits each. Compared with other encryption algorithms our proposed method works better for larger files similar to AES, whereas RSA takes lots of time out of all next to it is Triple DES and then comes DES. All encryption algorithms take less time for Decryption than Encryption, RSA takes highest time for Decryption and Blowfish takes lowest time for Decryption and AES-Scrypt is similar to AES in this case also where it takes a little more time than AES as it should process two keys again to get the hash for decryption. Coming to memory used RSA consumes more memory compared to all the algorithms and Blowfish uses the least per unit operations.Where as memory usage is high but not higher than RSA in AES-Scrypt algorithm as it is processing two keys which are of each 128 bits.Coming to complexity of an algorithm RSA and Blowfish shows Military Grade security where DES have low security as it uses 40 to 50 bit shared key its standard of complexity is Very low. Security Levels are pretty high for RSA and Blowfish and least for DES but AES-Scrypt provides pretty good Security levels compared to RSA.

Table:1



	AES-Scrypt	RSA	Blowfish	Triple DES	AES	DES
Encryption Time	Similar to AES	Very High	Low	High	Moderate	Comparatively high with AES
Decryption Time	Similar to AES	Very High	Low	High	Moderate	Comparatively High with AES
Memory Usage	High compared to Triple DES	Very High	Low	High compared to AES but lower than RSA	Moderate	High compared with AES but lower than Triple DES
Complexity	Very high compared to AES and Triple DES	Very High	High	High	Comparatively higher than DES and Triple DES	Low
Security	Very High compared with Triple DES	Military Grade	Military Grade	Very High	High	Low

IX. Conclusion

Even the AES-Scrypt provides nearly Military grade security levels it uses comparatively high memory space which we can consider as one minor drawback which can be negligible with current technology in the hands of human kind.There may be further Developments also for this technique in order to improve the efficiency of the algorithm and reduce disk consumption. This algorithm can be used where there are two private parties ,want to exchange information or collectively want to secure common information. The main advantage of this method is the attacker cannot break even one key as the Scrypt makes it hard to compute the hash of the master key itself which is generated from two private keys which makes it impossible for the attacker to crack even a single password. Even if the attacker broke a password he need to compute 2^{128} iterations to crack a single password after cracking its hash which makes it even more complex.

X. ACKNOWLEDGEMENT

We are thankful to our sir Dr. SHANTHARAJAH S P for providing us an opportunity to work on this project and encouraging us to do our best during the project.

XI. REFERENCES

[1] P.P. Dang; P.M. Chau, Image encryption for secure Internet multimedia applications, IEEE, Aug 2000

[2] O. Y. H. Cheung, K. H. Tsoi, P. H. W. Leong,M. P. Leong ,Tradeoffs in Parallel and Serial Implementations of the International Data Encryption Algorithm IDEA, Springer, 2001

[3] Nithin N , Anupkumar M Bongale , G. P. Hegde, Image Encryption based on FEAL algorithm, citeseerx

[4] P. Hamalainen; M. Hannikainen; T. Hamalainen; J. Saarinen, Configurable hardware implementation of triple-DES encryption algorithm for wireless local area network, IEEE, 2001

[5] Zhou Yingbing; Li Yongzhen, The design and implementation of a symmetric encryption algorithm based on DES, IEEE, 2014

[6] Daniar Heri Kurniawan; Rinaldi Munir, Double
Chaining Algorithm: A secure symmetric-key encryption algorithm, IEEE, 2016

[7] Babitha M.P.; K.R. Remesh Babu ,Secure cloud storage using AES encryption, IEEE, 2016

[8] Noura Aleisa ,A Comparison of the 3DES and AES Encryption Standards, International Journal of Security and Its Applications, 2014

[9] Suli Wang; Ganlai Liu ,File Encryption and Decryption System Based on RSA Algorithm, IEEE, 2011

[10] Priyadarshini Patila,*, Prashant Narayankarb ,Narayan D G c , Meena S Md ,A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish ,ELSEVIER, 2015

[11] Xiaoyun Wang,Hongbo Yu, How to Break MD5 and Other Hash Functions, Springer, 2005

[12] Friedrich Wiemer; Ralf Zimmermann, High-speed implementation of bcrypt password search using special-purpose hardware, IEEE, 2014

[13] Henri Gilbert,Helena Handschuh, Security Analysis of SHA-256 and Sisters, Springer, 2003

[14] Shay Gueron; Simon Johnson; Jesse Walker, SHA-512/256, IEEE, 2011

[15]C. Percival, S. Josefsson, The scrypt Password-Based Key Derivation Function,Internet Engineering Task Force (IETF), 2016

[16] Anne Barsuhn, Stefan Lucks, Christian Forler Bauhaus-Universit¨at Weimar, Cache Timing Attack on Scrypt, BOSCH 19th Crypto Day, 2013

[17]Eran Tromer, Dag Arne Osvik , Adi Shamir,Efficient Cache Attacks on AES, and Countermeasures, Journal of Cryptology, 2010

[18]Joël Alwen,Binyi Chen,Krzysztof Pietrzak,Leonid Reyzin,Scrypt Is Maximally Memory-Hard, Springer,2017