Literature Review

Securing file storage using hybrid encryption and cryptology

Tushar Agarwal
1783910056
Agarwal.tushar2905@gmail.com

Shivendra Trivedi
1783910051
Shivendratrivedi2014@gmail.com

Abstract— Now a day's cloud computing is used in many areas like industry, military colleges etc to storing huge amount of data. We can retrieve data from cloud on request of user. To
store data on cloud we have to face many issues. To provide the solution to these issues there are n number of ways Cryptography and steganography techniques are more popular
now a day's for data security. Use of a single algorithm is not effective for high level security to data in cloud computing. In this paper we have introduced new security mechanism using
symmetric key cryptography algorithm and steganography. In this proposed system AES, blowfish, RC6 and BRA algorithms are used to provide block wise security to data. All algorithm key size is 128 bits. LSB steganography technique is introduced for key
information security. Key information contains which part of file is encrypted using by which algorithm and key. File is splited into eight parts. Each and every part of file is encrypted using
different algorithm. All parts of file are encrypted simultaneously with the help of multithreading technique. Data encryption Keys are inserted into cover image using LSB technique. Stego image is send to valid receiver using email .For file decryption purpose reverse process of encryption is applied. All parts of file are encrypted simultaneously with the help of multithreading technique. Data encryption Keys are inserted into cover image using LSB technique. Stego image is send to valid receiver using email .For file decryption purpose reverse process of encryption is applied. Cloud security is defensive method to protect data and there

are various methods to protect data like Deterrent controls, Preventive controls, corrective controls and detective controls. With concern of security we should keep some points in mind like privacy, confidentiality, integrity and so on. And our novel research based on this.

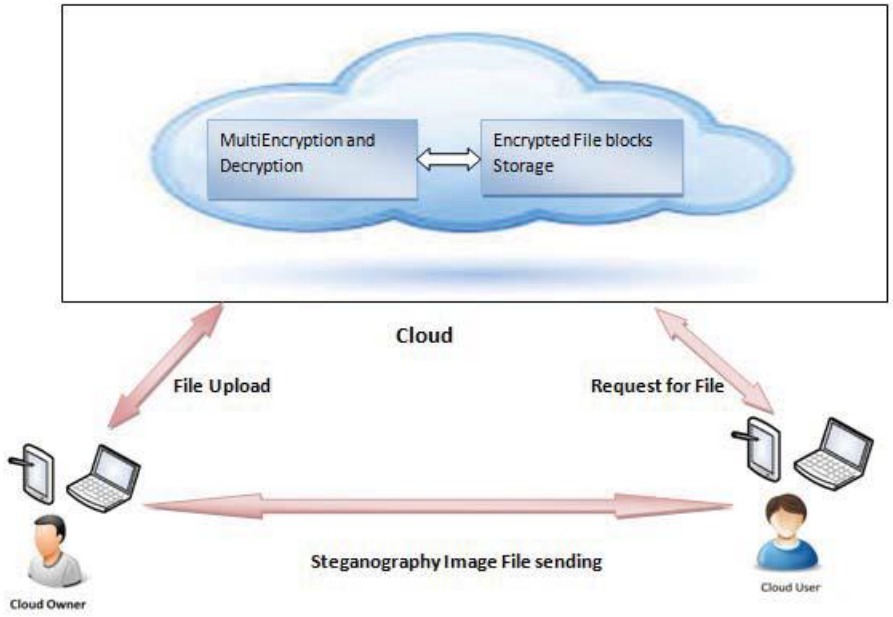
Cloud computing is originated from earlier large-scale distributed computing technology. NIST defines Cloud computing as a model for enabling convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". In Cloud computing, both files and software are not fully contained on the user's computer. File security concerns arise because both user's application and program are residing in provider premises. The cloud provider can solve this problem by encrypting the files by using encryption algorithm. This paper presents a file security model to provide an efficient solution for the basic problem of security in cloud environment. In this model, hybrid encryption is used where files are encrypted by blowfish coupled with file splitting and SRNN (modified RSA) is used for the secured communication between users and the servers

Introduction

Cryptography technique translates original data into unreadable form. Cryptography technique is divided into symmetric key cryptography and public key cryptography. This technique uses keys for translate data into unreadable form. So only authorized person can access data from cloud server. Cipher text data is visible for all people. Symmetric key cryptography algorithms are AES, DES,3DES, IDEA, BRA and blowfish. The main issue is delivering the key to receiver into multi user application. These algorithms require low delay for data encode decode but provides low security. Public key cryptography algorithm is RSA and ECC algorithm. Public and private keys are manipulated into public key cryptography algorithms. These algorithms accomplished high level security but increase delay for data encode and decode. Steganography hide the secret data existence into envelope. In this technique existence of data is not visible to all

people. Only valid receiver knows about the data existence. Text steganography technique is used to produce high security for data. Secret data of user hide into text cover file. After adding text into text cover file, it looks like normal text file. If text file found by illegitimate user than
also, cannot get sensitive data. If illegitimate user tries to recover original data than large amount of time is essential. DES algorithm is used for text encode and decode. Advantage of text steganography technique is providing security to text. Minimum space is essential for text steganography as compare to image steganography Three-bit LSB technique used for image steganography. This system is suggested by author R.T. Patil. Sensitive data of user hide into cover image. We can hide huge amount of into image using LSB steganography technique. The author Klaus Hafmann has implemented high throughput architecture for cryptography algorithm. AES is symmetric key cryptography algorithm. It supports three types of keys. For 128-bit key require 10 rounds,192-bit key require 12 rounds and 256-bit key require 14 rounds. In improved AES algorithm encryption and decryption time is reduced. Advantage of modified AES algorithm is providing better performance in terms of delay New symmetric key cryptography algorithm is presented by author M. Nagle. It applies a single key for texts encode
and decode. Size of key is 128 bits. In this algorithm many steps are executed randomly so illegitimate user can even guess the steps of algorithm. Provide high throughput is one of the advantage of symmetric key cryptography algorithms. Improved DES algorithm uses 112-bit key size for data encode and decode. For data encode purpose two keys are used.128-bit input of DES algorithm is divided into two parts. That two parts are executed at a same time. DES algorithm has one weakness. That is less key size.3DES algorithm essential large amount of time for encryption and decryption. Improved DES algorithm has capability of provide better performance as compare to DES and 3DES. Name Based Encryption Algorithm is work on one byte at a time. It uses secret key for encryption and decryption. Key generation process is done using random key generation technique. It provides security to data. Disadvantage of this algorithm is essential maximum time for converting data into cipher text because it operates on single byte at a time. To solve data storage and security issues author has new security model. In this model private and public cloud storage areas are used for increase security level of data. On private cloud secure data is stored and unnecessary data

is stored on public cloud. Because public text cover file. After adding text into text cover file, it looks like normal text file. If text file found by illegitimate user than also cannot get sensitive data. If illegitimate user tries to recover original data than large amount of time is essential. DES algorithm is used for text encode and decode. Advantage of text steganography technique is providing security to text. Minimum space is essential for text steganography as compare
to image steganography. Three-bit LSB technique used for image steganography. This system is suggested by author R.T. Patil. Sensitive data of user hide into cover image. We can hide huge amount of into image using LSB steganography technique. The author Klaus Hafmann has implemented high throughput architecture for cryptography algorithm. AES is symmetric key cryptography algorithm. It supports three types of keys.
. To solve above issues we have introduced new security mechanism.



Cloud owner and cloud user are included into system architecture as show in above figure Cloud owner upload the data on cloud server. File is split into octet. Every part of file is encoded simultaneously using multithreading technique. Encoded file is stored on cloud server. Keys used for encryption are stored into cover image. Cloud computing is the multi user environment. In this more than one user can access file from cloud server. Cloud user request for file. On request of file user

also get stego image using email which consist of key information. Reverse process is used for decode the file.

Data Security Issues

Due to openness and multi-tenant characteristics of the cloud, the traditional security mechanisms are no longer suitable for applications and data in cloud. Some of the issues are as following:

- Due to dynamic scalability, service and location transparency features of cloud computing model, all kinds of application and data of the cloud platform have no fixed infrastructure and security boundaries. In the event of security breach, it is difficult to isolate a particular resource that has a threat or has been compromised.
- According to service delivery models of Cloud computing, resources and cloud services may be owned by multiple providers. As there is a conflict of interest, it is difficult to deploy a unified security measure.
- Due to the openness of cloud and sharing virtualized resources by multitenant, user data may be accessed by other unauthorized users.

Hybrid Cryptosystem Scheme

In order to ensure file security on cloud, hybrid cryptosystem is being used. We assume that the remote server is trusted, so files are encrypted by server and finally encrypted files are stored at the server end. The hybrid cryptosystem uses a combination of:

- ChaCha20 Algorithm
- Poly1305 Algorithm

.

CHACHA20

The description of the ChaCha algorithm will at various times refer to the ChaCha state as a "vector" or as a "matrix". This follows the use of these terms in Professor Bernstein's paper. The matrix notation is more visually convenient and gives a better notion as to why some rounds are called "column rounds" while others are called "diagonal rounds". Here's a diagram of how the matrices relate to vectors (using the C language convention of zero being the index origin).

```
 0  1  2  3
 4  5  6  7
 8  9 10 11
12 13 14 15
```

The elements in this vector or matrix are 32-bit unsigned integers. The algorithm name is "ChaCha". "ChaCha20" is a specific instance where 20 "rounds" (or 80 quarter rounds) are used.

The ChaCha Quarter Round

The basic operation of the ChaCha algorithm is the quarter round.It operates on four 32-bit unsigned integers, denoted a, b, c, and d.The operation is as follows (in C-like notation):

1. a += b; d ^= a; d <<<= 16;
2. c += d; b ^= c; b <<<= 12;
3. a += b; d ^= a; d <<<= 8;
4. c += d; b ^= c; b <<<= 7;

here "+" denotes integer addition modulo 2^32, "^" denotes a bitwise Exclusive OR (XOR), and "<<< n" denotes an n-bit left rotation (towards the high bits).

For example, let's see the add, XOR, and roll operations from the fourth line with sample numbers:

- a = 0x11111111
- b = 0x01020304
- c = 0x77777777
- d = 0x01234567
- c = c + d = 0x77777777 + 0x01234567 = 0x789abcde
- b = b ^ c = 0x01020304 ^ 0x789abcde = 0x7998bfda
- b = b <<< 7 = 0x7998bfda <<< 7 = 0xcc5fed3c

Test Vector for the ChaCha Quarter Round

For a test vector, we will use the same numbers as in the example, adding something random for c.

- a = 0x11111111
- b = 0x01020304
- c = 0x9b8d6f43
- d = 0x01234567

After running a Quarter Round on these four numbers, we get these:

- a = 0xea2a92f4
- b = 0xcb1cf8ce
- c = 0x4581472e
- d = 0x5881c4bb

A Quarter Round on the ChaCha State

The ChaCha state does not have four integer numbers: it has 16. So the quarter-round operation works on only four of them -- hence the name. Each quarter round operates on four predetermined numbers in the ChaCha state. We will denote by QUARTERROUND(x,y,z,w) a quarter- round operation on the numbers at indices x, y, z, and w of the ChaCha states when viewed as a vector. For example, if we apply QUARTERROUND (1,5,9,13) to a state, this means running the quarter-round operation on the elements marked with an asterisk, while leaving the others alone:

```
 0  *a   2   3
 4  *b   6   7
 8  *c  10  11
12  *d  14  15
```

Note that this run of quarter round is part of what is called a "column round".

Test Vector for the Quarter Round on the ChaCha State

For a test vector, we will use a ChaCha state that was generated randomly:

Sample ChaCha State

```
879531e0  c5ecf37d  516461b1  c9a62f8a
44c20ef3  3390af7f  d9fc690b  2a5f714c
53372767  b00a5631  974c541a  359e9963
5c971061  3d631689  2098d9d6  91dbd320
```

We will apply the QUARTERROUND(2,7,8,13) operation to this state. For obvious reasons, this one is part of what is called a "diagonal round":

After applying QUARTERROUND(2,7,8,13)

```
879531e0  c5ecf37d *bdb886dc  c9a62f8a
44c20ef3  3390af7f  d9fc690b *cfacafd2
*e46bea80  b00a5631  974c541a  359e9963
5c971061 *ccc07c79  2098d9d6  91dbd320
```

Note that only the numbers in positions 2, 7, 8, and 13 changed.

The ChaCha20 Block Function

The ChaCha block function transforms a ChaCha state by running multiple quarter rounds.
The inputs to ChaCha20 are:

1. A 256-bit key, treated as a concatenation of *eight 32-bit* little-endian integers.
2. A 96-bit nonce, treated as a concatenation of *three 32-bit* little-endian integers.
3. A 32-bit block count parameter, treated as a *32-bit* little-endian integer.

The output is 64 random-looking bytes.T**he ChaCha algorithm described here uses a 256-bit key.**

Modification in Algorithm

.

We have modified this here to be more consistent with recommendations in . This limits the use of a single (key,nonce) combination to 2^32, but that is enough for most uses. In cases where a single key is used by multiple senders, it is important to make sure that they don't use

the same nonces. Recommended Nonce Formation

The following method to construct nonces is RECOMMENDED. The nonce is formatted as illustrated in Figure 1, with the initial octets consisting of a Fixed field, and the final octets consisting of a Counter field. For each fixed key, the length of each of these fields, and thus the length of the nonce, is fixed. Implementations SHOULD support 12-octet nonces in which the Counter field is four

octets long.

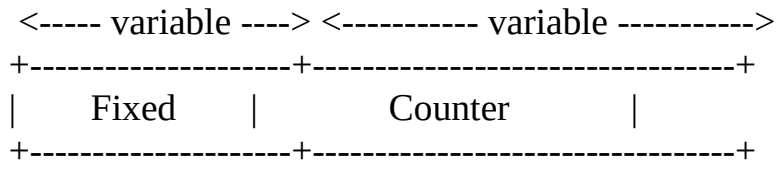


Figure 1: Recommended nonce format

The Counter fields of successive nonces form a monotonically increasing sequence, when those fields are regarded as unsigned integers in network byte order. The length of the Counter field MUST remain constant for all nonces that are generated for a given encryption device. The Counter part SHOULD be equal to zero for the first nonce, and increment by one for each successive nonce that is

generated. However, any particular Counter value MAY be skipped over, and left out of the sequence of values that are used, if it is convenient. For example, an application could choose to skip the initial Counter=0 value, and set the Counter field of the initial nonce to 1. Thus, at most 2^(8*C) nonces can be generated when the Counter field is C octets in length. In some cases, it is desirable to not transmit or store an entire nonce, but instead to reconstruct that value from contextual information immediately prior to decryption. As an example, ciphertexts could be stored in sequence on a disk, and the nonce for a particular ciphertext could be inferred from its location, as long as the rule for generating the nonces is known by the decrypter. We call the portion of the nonce that is stored or sent with the ciphertext the explicit part.
This can be assured by partitioning the nonce space so that the first 32 bits are unique per sender, while the other 64 bits come from a counter.

The ChaCha20 state is initialized as follows:

The first four words (0-3) are constants: 0x61707865, 0x3320646e, 0x79622d32, 0x6b206574.

The next eight words (4-11) are taken from the 256-bit key by reading the bytes in little-endian order, in 4-byte chunks.

Word 12 is a block counter. Since each block is 64-byte, a 32-bit word is enough for 256 gigabytes of data.

Words 13-15 are a nonce, which should not be repeated for the same key. The 13th word is the first 32 bits of the input nonce taken as a little-endian integer, while the 15th word is the last 32 bits.

```
cccccccc  cccccccc  cccccccc  cccccccc
kkkkkkkk  kkkkkkkk  kkkkkkkk  kkkkkkkk
kkkkkkkk  kkkkkkkk  kkkkkkkk  kkkkkkkk
bbbbbbbb  nnnnnnnn  nnnnnnnn  nnnnnnnn
```

c=constant k=key b=blockcount n=nonce

ChaCha20 runs 20 rounds, alternating between "column rounds" and "diagonal rounds". Each round consists of four quarter-rounds, and they are run as follows. Quarter rounds 1-4 are part of a "column"
round, while 5-8 are part of a "diagonal" round:

1. QUARTERROUND (0, 4, 8,12)
2. QUARTERROUND (1, 5, 9,13)
3. QUARTERROUND (2, 6,10,14)
4. QUARTERROUND (3, 7,11,15)
5. QUARTERROUND (0, 5,10,15)
6. QUARTERROUND (1, 6,11,12)
7. QUARTERROUND (2, 7, 8,13)
8. QUARTERROUND (3, 4, 9,14)

At the end of 20 rounds (or 10 iterations of the above list), we add the original input words to the output words, and serialize theresult by sequencing the words one-by-one in little-endian order.

The Poly1305 Algorithm

Poly1305 is a one-time authenticator designed by D. J. Bernstein. Poly1305 takes a 32-byte one-time key and a message and produces a 16-byte tag. This tag is used to authenticate the message.

The original article is titled "The Poly1305-AES message-authentication code", and the MAC function there requires a 128-bit AES key, a 128-bit "additional key", and a 128-bit (non-secret) nonce. AES is used there for encrypting the nonce, so as to get a unique (and secret) 128-bit string, but as the paper states, "There is nothing special about AES here. One can replace AES with an arbitrary keyed function from an arbitrary set of nonces to 16-byte strings."

Regardless of how the key is generated, the key is partitioned into two parts, called "r" and "s". The pair (r,s) should be unique, and MUST be unpredictable for each invocation (that is why it was originally obtained by encrypting a nonce), while "r" MAY be constant, but needs to be modified as follows before being used: ("r" is treated as a 16-octet little-endian number):

- r[3], r[7], r[11], and r[15] are required to have their top four bits clear (be smaller than 16)
- r[4], r[8], and r[12] are required to have their bottom two bits clear (be divisible by 4)

The following sample code clamps "r" to be appropriate:
#include "poly1305aes_test.h"
void poly1305aes_test_clamp(unsigned char r[16])
{r[3] &= 15; r[7] &= 15; r[11] &= 15; r[15] &= 15; r[4] &= 252; r[8] &= 252;
br[12] &= 252; }

The "s" should be unpredictable, but it is perfectly acceptable to generate both "r" and "s" uniquely each time. Because each of them is 128 bits, pseudorandomly generating them is alsobacceptable.

The inputs to Poly1305 are:

- A 256-bit one-time key
- An arbitrary length message

The output is a 128-bit tag. First, the "r" value should be clamped.
Next, set the constant prime "P" be 2^130-5: 3ffffffffffffffffffffffffffffffb. Also set a variable "accumulator" to zero.

Next, divide the message into 16-byte blocks. The last one might be shorter:

- Read the block as a little-endian number.
- Add one bit beyond the number of octets. For a 16-byte block, this is equivalent to adding 2^128 to the number. For the shorter block, it can be 2^120, 2^112, or any power of two that is evenly divisible by 8, all the way down to 2^8.
- If the block is not 17 bytes long (the last block), pad it with zeros. This is meaningless if you are treating the blocks as numbers.
- Add this number to the accumulator.
- Multiply by "r".
- Set the accumulator to the result modulo p. To summarize: Acc =((Acc+block)*r) % p.

Finally, the value of the secret key "s" is added to the accumulator, and the 128 least significant bits are serialized in little-endian order to form the tag.

AEAD Construction

AEAD_CHACHA20_POLY1305 is an authenticated encryption with additional data algorithm. The inputs to AEAD_CHACHA20_POLY1305 are:

- A 256-bit key
- A 96-bit nonce -- different for each invocation with the same key

- An arbitrary length plaintext
- Arbitrary length additional authenticated data (AAD)

Some protocols may have unique per-invocation inputs that are not 96 bits in length. For example, IPsec may specify a 64-bit nonce.In such a case, it is up to the protocol document to define how to transform the protocol nonce into a 96-bit nonce, for example, by concatenating a constant value.

The ChaCha20 and Poly1305 primitives are combined into an AEAD that takes a 256-bit key and 96-bit nonce as follows:

- First, a Poly1305 one-time key is generated from the 256-bit key and nonce using the procedure.
- Next, the ChaCha20 encryption function is called to encrypt the plaintext, using the same key and nonce, and with the initial counter set to 1.
- Finally, the Poly1305 function is called with the Poly1305 key calculated above, and a message constructed as a concatenation of the following:

i. **The AAD**
ii. **padding1 --** the padding is up to 15 zero bytes, and it brings the total length so far to an integral multiple of 16. If the length of the AAD was already an integral multiple of 16 bytes, this field is zero-length.
iii. **The ciphertext**
iv. **padding2 --** the padding is up to 15 zero bytes, and it brings the total length so far to an integral multiple of 16. If the length of the ciphertext was already an integral multiple of 16 bytes, this field is zero-length.
v. The length of the additional data in octets (as a 64-bit little-endian integer).
vi. The length of the ciphertext in octets (as a 64-bit little-endian integer).

The output from the AEAD is twofold:

- A ciphertext of the same length as the plaintext.
- A 128-bit tag, which is the output of the Poly1305 function.

Decryption is similar with the following differences:

- The roles of ciphertext and plaintext are reversed, so the ChaCha20 encryption function is applied to the ciphertext, producing the plaintext.
- The Poly1305 function is still run on the AAD and the ciphertext, not the plaintext.

- The calculated tag is bitwise compared to the received tag. The message is authenticated if and only if the tags match.

A few notes about this design:

1. The amount of encrypted data possible in a single invocation is 2^32-1 blocks of 64 bytes each, because of the size of the block counter field in the ChaCha20 block function. This gives a total of 247,877,906,880 bytes, or nearly 256 GB. This should be enough for traffic protocols such as IPsec and TLS, but may be too small for file and/or disk encryption. For such uses, we can return to the original design, reduce the nonce to 64 bits, and use the integer at position 13 as the top 32 bits of a 64-bit block counter, increasing the total message size to over a million petabytes (1,180,591,620,717,411,303,360 bytes to be exact).
2. Despite the previous item, the ciphertext length field in the construction of the buffer on which Poly1305 runs limits the ciphertext (and hence, the plaintext) size to 2^64 bytes, or sixteen thousand petabytes (18,446,744,073,709,551,616 bytes to be exact).

The AEAD construction in this section is a novel composition of ChaCha20 and Poly1305. A security analysis of this composition is given in Here is a list of the parameters for this construction as defined in.

- K_LEN (key length) is 32 octets.
- P_MAX (maximum size of the plaintext) is 247,877,906,880 bytes, or nearly 256 GB.
- A_MAX (maximum size of the associated data) is set to 2^64-1 octets by the length field for associated data.
- N_MIN = N_MAX = 12 octets.
- C_MAX = P_MAX + tag length = 247,877,906,896 octets.

Distinct AAD inputs shall be concatenated into a single input to AEAD_CHACHA20_POLY1305. It is up to the application to create a structure in the AAD input if it is needed.

Our Proposed System

The fundamentals of the ChaCha20 have been studied to build a new keystream generator for producing keys, taking into consideration the increasing security level and decreasing the complicated stages. The producing keys will be used for encryption of IoT data. We denote the new approach as Super ChaCha Lightweight Stream Cipher, described in Algorithm 1, which consists of 10 rounds ultimately summarized into:

1. The rotation procedure (16, 12, 8, and 7) in ChaCha20 is modified from a fixed constant to a variable constant based on random value (y0, y1, y2, and y3), respectively, in each round, as shown in Algorithm
2. The order of application of the QRF (for updating inputs) has been changed in the columns from followed by diagonal form to zigzag form and then by alternate form as shown in Figs. 1 and 2. This new order of updating process results in more diffusion of inputs and thereby increasing the complexity against attacks.

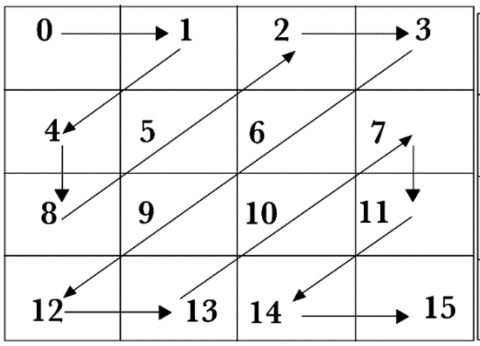


Figure 1

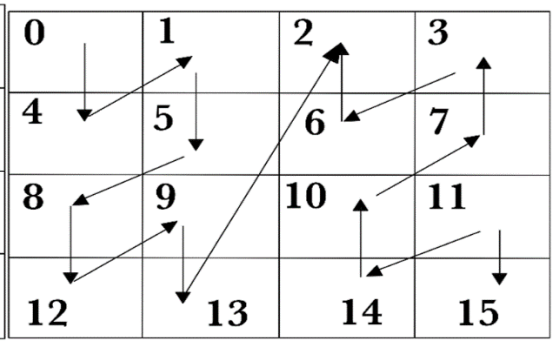


Figure 2

Algorithm

Algorithm 1: Super ChaCha keystream Generator
Input: 512 bit (seed key, nonce, counter and constants) stored in a matrix x (x0,x1, x2,…….,x15)
Output: 512 bit (keystream)
Begin
Step1: round=1
Step2: repeat
ChaCha-quarter function (x0, x1, x 4, x 8)
ChaCha-quarter function (x5, x2, x3, x6)
ChaCha-quarter function (x9, x12, x13, x10)
ChaCha-quarter function (x7, x 11, x 14, x 15).
ChaCha-quarter function (x0, x4 x1, x5),
ChaCha-quarter function (x8, x12, x9,x13),
ChaCha-quarter function (x2, x6, x3, x7),
ChaCha-quarter function (x 10, x 14, x11, x15).
Step3: round =round+1
Step4: until round less than or equal to 10
Step5: store the update matrix x in keystream matrix as key
End

Algorithm 2:
ChaCha-quarter function Input: Four 32-bit (a, b, c, d)
Output: Update Four 32-bit (a, b, c, d).
Begin Step1: y0=first 4 bits of c , y1=first 4 bits of a,y2=first 4 bits of b,y3=first 4 bits of d) //// Range of y0,y1,y2 and y3 from 0 to 15
Step2: a = a + b d=(d ⊕ a) <<< y0) c = c + d b=(b ⊕ c) <<< y1) a = a + b d=(d ⊕ a) <<< y2) c = c + d b=(b ⊕ c) <<< y3)
End

Power Consumption

1. The power consumption, of the suggested procedures executing on NodeMCU for the first installation, is computed by the following concepts:
2. Running Procedure in microseconds: It acts the entire time in microseconds that are needed for performing the suggested procedure
3. Cycles of procedure: It acts the value of running a procedure in microseconds * 80, due to the CPU clock performs at a frequency of the eighty MHz (eighty times each microsecond).

- As previously discussed in Node MCU properties, the voltage is 3.3 and the current is 0.08 A.
- Clock Cycles: It is equal to the division of 1 by frequency that is referring to $80 \times 10-$

4. The measure of power consumption has been calculated in microjoules by the product of the voltage, current, cycles of procedure The results show very small increase in energy consumption associated with using Super ChaCha.

Related Work

Hybrid cryptography algorithm present by author A. Shahade. AES and RSA algorithms are used into hybrid algorithm. AES algorithm require a single key. In hybrid algorithm three keys are used. For data upload on cloud mandatory keys are AES secret key and RSA public key.
Private key of RSA and AES secret key are essential to download data from cloud. Whenever use makes an effort to upload data on cloud first that file stored onto directory for
short time. In encryption process first AES algorithm is applied on file after that RSA algorithm is applied on encrypted data. Reverse process is followed for decryption. After applying keys that file covert into encoded form and stored on cloud server. Advantages of hybrid algorithm are
data integrity, security, confidentiality and availability. Disadvantage of RSA algorithm is large amount time essential for data encode and decode.

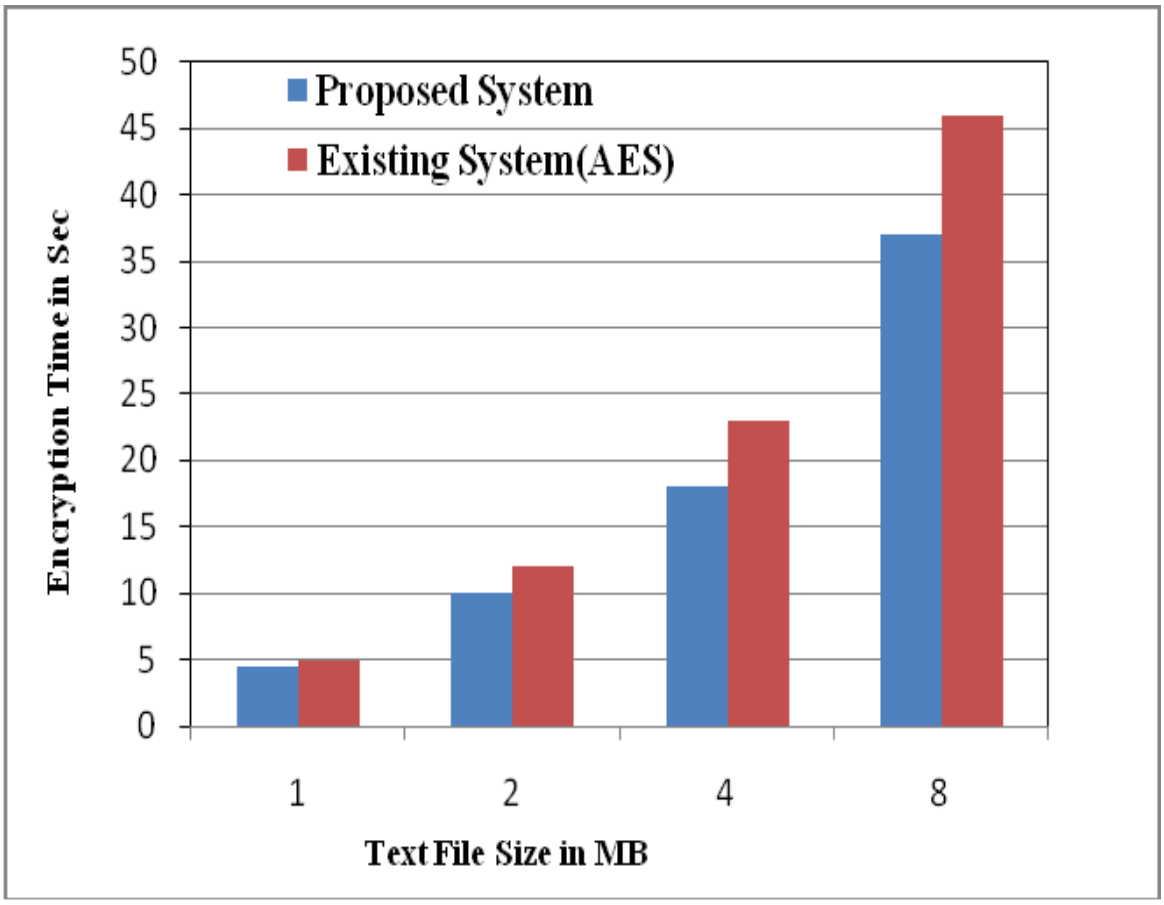
In security model symmetric algorithm uses chunk level encryption and decryption of data in cloud computing. Key size is 256 bits. Key is rotated to achieve high level security. For data integrity purpose hash value is generated. Hash values are garneted after encryption and before decryption. If both hash values match than that data is in correct form. In
this security model only valid user can access data from cloud. Advantages of security model are integrity, security and confidentiality.

Three algorithms are used for implementation of hybrid algorithm. User authentication purpose digital signature is used. Blowfish algorithm is used to produce high data

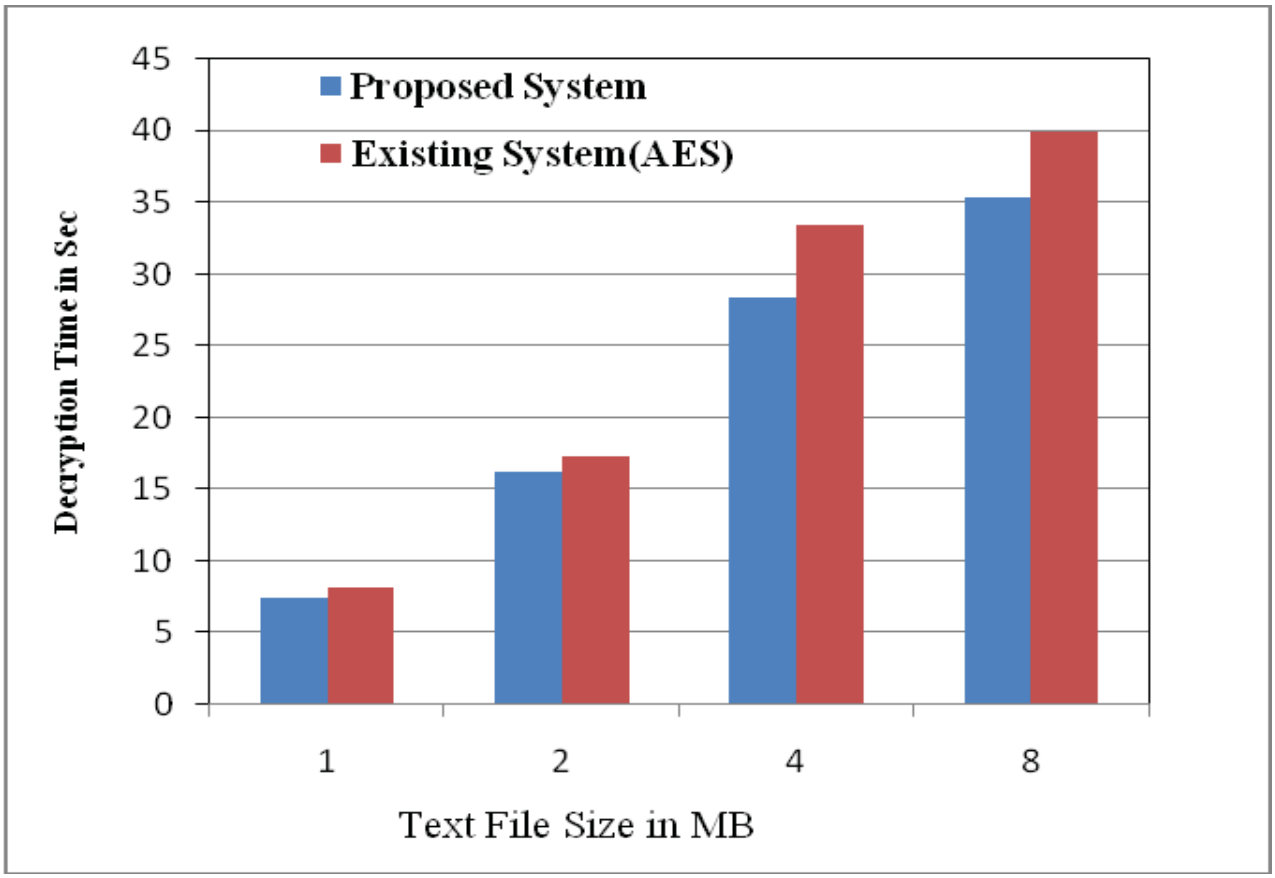
confidentiality. It is symmetric algorithm. It uses single key Blowfish algorithm need least amount of time for encode and decode. Sub key array concept is used into blowfish algorithm.
It is block level encryption algorithm. The main aim of this hybrid algorithm is achieved high security to data for upload and download from cloud. Hybrid algorithm solves the security, confidentiality and authentication issues of cloud.

Result Analysis

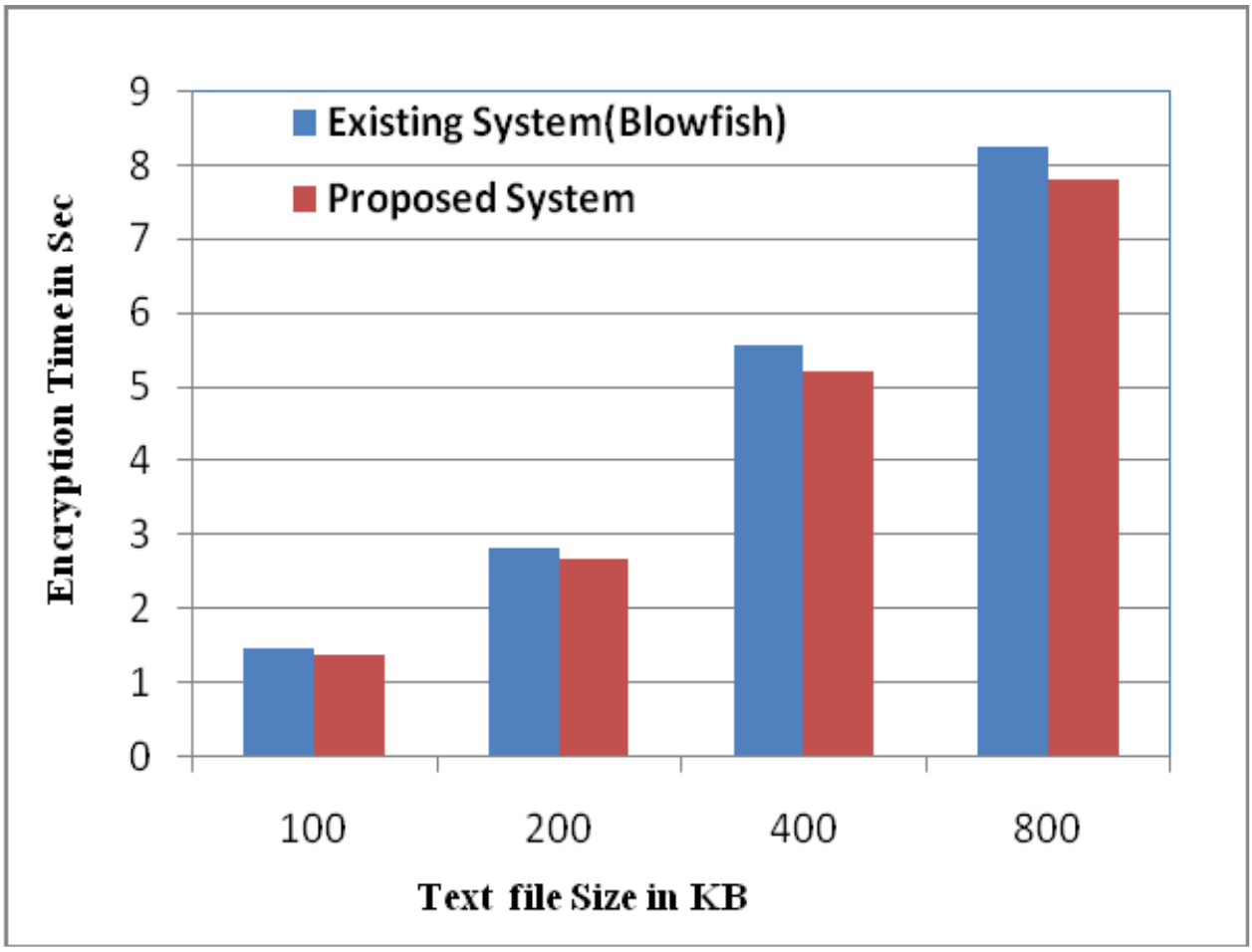
In this proposed system AES, RC6, Blowfish and BRA algorithms are used for block wise security to data. Proposed system is hybridization of AES, RC6, Blowfish and BRA. All
algorithms are symmetric key cryptography. These algorithms use a single key for file encode and decode purpose. All algorithms key size is 128 bits. To hide key information into
cover image using LSB technique. Implementation of proposed system is done using python language. File encoding and decoding time is calculated with the help of python programming. File encode and decode time is calculated for only text file with comparison of existing AES and Blowfish algorithms. File size is given in MB for AES algorithm. That is 1MB, 2MB, 4MB and 8MB.For Encode and decode time calculation of blowfish algorithm given file size is 100KB,200KB,400KB and 800KB.Encoding and decoding time is calculated in sec.



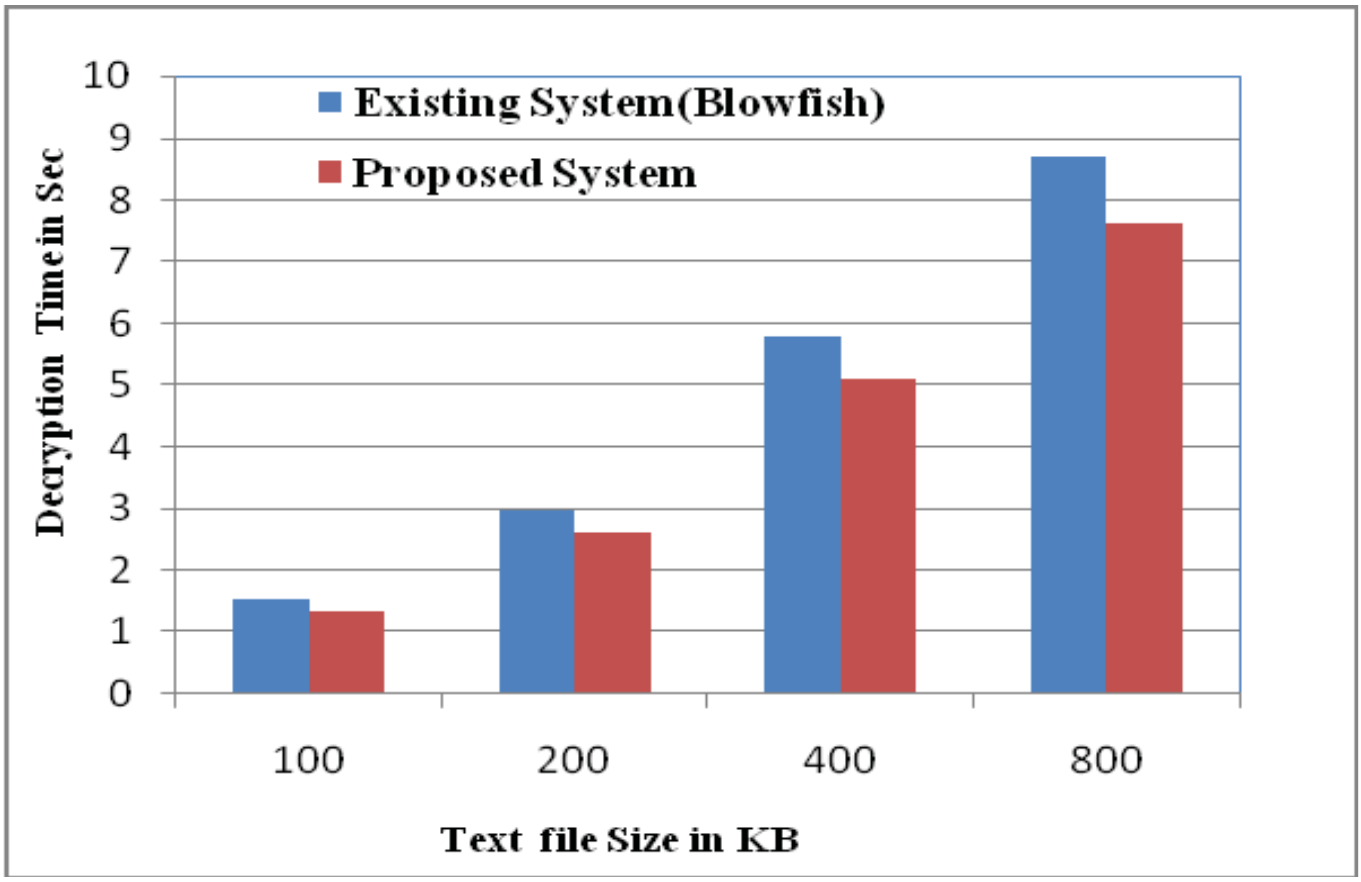
As shown in figure proposed system need least amount of time for file encode. Because in proposed system combination of symmetric key cryptography algorithms are run simultaneously. In Hybrid algorithm need 17% to 20% less time for text file as compare to Existing system. Use of single algorithm does not provide high level security to data in cloud computing...



As shown in above figure existing system need 15% to 17% maximum time need for file decryption purpose as compare to hybrid algorithm. AES algorithm is accomplished least amount of time for decryption. But provides less security to data. In AES if key size increases automatically number of rounds increases than encode and decode time also increases.



Blowfish need least amount of time for file encode with compare to Advance Encryption Standard algorithm. As shown in above figure proposed system 12% to 15% less time need for file encode as compare to Blowfish. In proposed hybrid algorithm uses a single key for data encode and decode.



In proposed system for text file decryption need 10% to 12% minimum time as compare to Blowfish as shown above figure in hybrid algorithm for file decryption needed maximum time as
compare to encryption. But Blowfish algorithm need minimum time for text file decode as compare to AES algorithm. For text file decryption needed maximum time in Blowfish algorithm as compare to encryption.

Hybrid Cryptosystem Phases

.

The hybrid cryptosystem used to maintain security of the files has two phases:

- Encryption Phase
- Decryption Phase

A. Encryption Phase

At the encryption end,

- On the specification of user, the file being encrypted will be sliced into n slices. Each of the file slices is encrypted using Blowfish key provided by the user for each slice.
- The key will be encrypted using SRNN public key
- After encryption, we have encrypted files slices and the corresponding encrypted keys.

B. Decryption Phase

At the decryption end,

- The user will provide n SRNN private keys, according to the number of slices (n) created during the encryption phase. Blowfish key is decrypted at the server end using the SRNN private key specific to the slice.
- Using the corresponding decrypted Blowfish keys, file slices stored at server are decrypted.
- The decrypted slices will be merged to generate original file.

Proposed Cloud Computing Security Architecture

In order to ensure file security on cloud, the above hybrid cryptosystem is deployed on cloud. We assume cloud server as trusted but in order to prevent tampering/misuse of data by intruder or data leakage or other security concerns, the data is stored at server in the encrypted form.
We broadly classify the scheme deployed on cloud in three phases:

- Registration Phase
- Uploading Phase
- Downloading Phase

We used Open Nebula toolkit to set up cloud environment. In Open Nebula, we have one front node and n cluster nodes. The VM's are deployed from front node to the corresponding cluster node. Open Nebula has been designed in such a way that it allows integration with many different hypervisors and environments. There is a front-end that executes all the process in Open-Nebula while the cluster nodes provide the resources that are needed by VM. There is at least one physical network joining all the cluster nodes with the frontend.

A. Registration Phase

In the Registration Phase, the client registers himself in order to upload and view his files to/from the cloud server. In the registration process, the client sends its request to front node and in return, front node assigns the VM of the cluster node, which has minimum load among other VM's on the network to the client. At the end of registration phase, the client is registered with IP address of corresponding VM. Whenever he again issues his request, the request is transferred to its corresponding VM. The encrypted files, encrypted blowfish keys, public SRNN keys are stored at his registered VM.

B. Uploading Phase

In the Uploading Phase, steps are as follows:

- Step: 1: The client will send request to front node to authenticate himself.
- Step 2: On successful authentication, the front end which send the corresponding IP address of the VM against which user was registered.
- Step 3: The files are uploaded by the client to the registered server (VM).
- Step 4: The encryption of uploaded files is done using the hybrid cryptosystem.
- Step 5: The encrypted slices and Blowfish encrypted keys remain stored in VM's data store.
- Step 6: The SRNN private keys are send to user and finally they are deleted form the server so that only the authenticated user is able to view his uploaded file.

C. Downloading Phase

In the downloading phase, the steps are as follows:

- Step 1: The client will send request to front node to authenticate himself.
- Step 2: On successful authentication, the front end which send the corresponding IP address of the VM against which user was registered
- Step 3: The client will upload n SRNN private keys for the corresponding n slices.
- Step 4: The SRNN private keys will decrypt the corresponding encrypted Blowfish keys and the encrypted slices are decrypted by Blowfish keys.
- Step 5: The decrypted files are merged to generate original file.
- Step 6: The decrypted file is downloaded and viewed at client end.

Design and Implementation

For the purpose of simulating the proposed cloud security model, we used Open Nebula open source toolkit. Here we created one front node and two cluster nodes. At each of the Cluster node 2 VM's are deployed. The allocation of VM at the time of registration is implemented in python which is well known for its platform independence. The hybrid cryptosystem is also implemented in python and deployed at each of the VM. Various libraries have been used like python. crypto. security to implement hybrid encryption scheme. The cloud security model has been tested for various types of file: audio, image, text, word, pdf file.

Benefits of Proposed Model

The proposed model is liable to meet the required security needs of data center of cloud. Blowfish used for the encryption of file slices takes minimum time and has maximum throughput for encryption and decryption from other symmetric algorithms. Modified RSA(SRNN) has increased security than RSA. The idea of splitting and merging adds on to meet the principle of data security. The hybrid approach when deployed in cloud environment makes the remote server more secure and thus, helps the cloud providers to fetch more trust of their users. For data security and privacy protection issues, the fundamental challenge of separation of sensitive data and access control is fulfilled. The various benefits are as summarized:

- The public key cryptography used helps to facilitate authorization of user for each file.
- The need of more light and secure encryption system for file information preserving system on cloud is satisfied.
- The file splitting and merging makes the model unfeasible to get attacked.

Conclusion and Future Work

Cloud storage issues are solved using cryptography and stenography techniques... Block wise Data security is achieved using AES, RC6, Blowfish and BRA algorithms.
Key information security is accomplished using LSB technique. Data integrity is accomplished using SHA1 hash algorithm. Low delay parameter is achieved using multithreading technique. With the help of proposed security mechanism data integrity, high security, low delay,
authentication and confidentiality parameters are accomplished. Using proposed Text file encryption need 17% to 20% less time as compare to AES algorithm. For AES text decryption needs 15% to 17% maximum time as compare to proposed system. In Blowfish for encryption need 12% to 15% maximum time as compare to proposed hybrid algorithm. Text
file decryption using hybrid algorithm need 10% to 12% less time with respect to Blowfish algorithm. In future, try to accomplish high level security using hybridization of public key cryptography algorithms.

According to service delivery models and deployment models of cloud, data security and privacy protection are the primary problems that need to be solved. Data Security and privacy issues exist in all levels in SPI service delivery models. The above-mentioned model is fruitful in data as a service, which can be extended in other service models of cloud. Also, it is tested in cloud environment like Open Nebula, in future this can be deployed in other cloud environments and the best among of all can be chosen.

Assumptions:

We assume that the remote server is trusted, so files are encrypted by server and finally encrypted files are stored at the server end. The hybrid cryptosystem uses a combination of chacha20 and poly1305.