**Final Report**

***In the Partial fulfilment of B. Tech. - III year***

**Efficient way of data retrieval from cloud using Blowfish Algorithm**

**Submitted by:-**

Manideep Pilli

1700153C203

CSE -2

**Submitted To: -**

Dr. Purnendu Shekhar Pandey

Assistant Professor

School of Engineering & Technology

Bml Munjal University



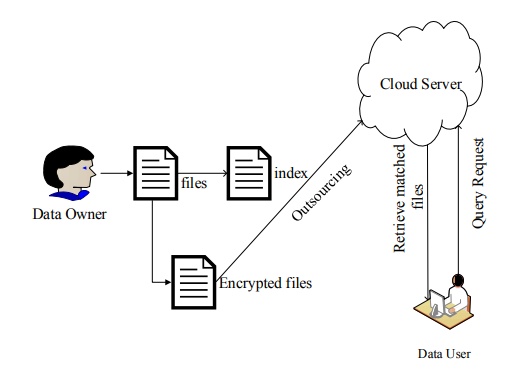
**Abstract**

The main issue with cloud environments, resources is privacy as they are shared among many servers, users and individuals. The role of privacy is that the data owner can upload the file and give access to only authorized people. It has been difficult for cloud providers to ensure file security. As this results in misuse of files, Data being modified, or data lost by intruders.So to ensure privacy, it requires strong file security. This report presents a security model which uses the concept of Blowfish encryption and decryption of the file which provides maximum security to files in the cloud.

**Introduction**

Nowadays there is millions of data storage and used, to reduce the cost of storage cloud computing is used. These Clouds should be secure about the privacy of the users, to ensure this security the data in clouds is encrypted

1. Generation of an index of data using Porter stemming.
2. Encryption of data and index using Blowfish encryption algorithm
3. The approved client produces catchphrase Query demand
4. After accepting the query demand from the client, utilizing the keyword search algorithm the coordinating files are searched.
5. The files are gotten to the client after decryption utilizing the Blowfish algorithm.



**Literature survey**

As the privacy of data in the cloud is danger suggested method uses the Blowfish algorithm for encryption and decryption designed in 1993 by Bruce Schneier, Blowfish has been providing good encryption rates and no powerful cryptanalysis of it has been found to date and proved more efficient than AES algorithm. In this model, the encryption will be done in 16 rounds and at last, the ciphertext is provided. The subkeys are generated for each round separately. These keys and ciphertext goes to a function f which further modifies the data. Blowfish encryption has not been broken till date.

**Blowfish Algorithm**

Blowfish Algorithm is encryption and decryption technique was made to be another option to DES algorithm and it is quicker than DES algorithm. Blowfish algorithm consists of a block size of 64-bit, 16 number of rounds,keysize from 32bits to 448bits,

4 substitution boxes and 18 subkeys stored in P array in form hexadecimal value of pi.

It goes through three different stages

1. KeyExpansion
2. Encryption
3. Decryption

**KeyExpansion**

The keys should be generated before proceeding to any encryption and decryption.

The p-array gas of 18 subkeys denoted with P0,P1,...P17, each of 32 bit size

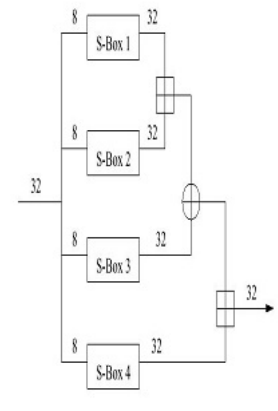
And four 32bit Substitution Boxes each of 256 hexadecimal entries, from

S1 [0] ,S1 [1] to S1 [255]

S2 [0] ,S2 [1] to S2 [255]

S3 [0] ,S3 [1] to S3 [255]

S4 [0] ,S4 [1] to S4 [255]



**Generation of Subkeys:**

1. Each of the subkeys is changed as for input key provided as:

1st element in P array is xor with 1st 32bit of key

2nd element in P array is xor with 2nd 32bit of key

.

.

.

i th element in the P array is xor with i th 32bit key and stored.

2.Initialise Substitution Boxes with hexadecimal values.

**CODE**

**def f(self,plaintext):** // Function F

**list=[0,0,0,0]**

**ans=""**

**for i in range(0,8,2):**

**col=int(self.hextobin(plaintxt[i:i+2]),2)** //hexadecimal to binary conversion

**a=int(i/2)**

**list[a]=self.s[a][col]** // appending the list with 4 S boxes values

**ans=self.addBin(list[0],list[1])** // Addition of two S boxes(Addition modulo for 2^32)

**ans=self.xor(ans,list[2])** // Xor for previous addition with with 3rd S box

**ans=self.addBin(ans,list[3])** // Addition of previous xor S boxes with 4 th S box

**return(ans)**

**def keygernation(self,key):**

**j=0**

**for i in range(len(self.p)):**

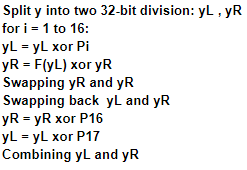
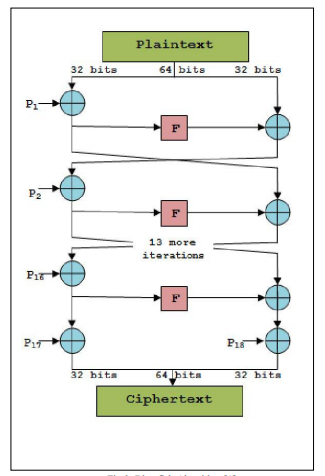
**self.p[i]=self.xor(self.p[i],key[j:j+8])** // xoring Key by dividing into 16 bits with P values

**# print("subkey"+str(i+1)+" : "+str(self.p[i]))**

**j=(j+8)%(len(key))** // for next xor using next 16bits

**Encryption**

It goes through a function repeatedly 16 times in the system. All tasks are XORs and added on 32bit words.

****

**CODE**

**def round(self,i,plaintext):**

**left=plaintext[:8]** // First 0 to 7 bits of plaintext

**right=plaintext[8:]** // 8 to 15 bits of plaintext

**left =self.xor(left, self.p[i])** // xor left with P value(0 to 17 values)

**if len(left)<8:**

**left="0" +left[:]**

**fOut=self.f(left)** // passing xored value to F function

**right =self.xor(fOut,right)** // xor fout with right(8 to 15 bits)

**if len(right)<8:**

**right ="0" +right[:]**

**return right + left** // returning the concatenated value of right and left which send as plain text for next round

**def encrypt(self,plaintext):**

**for i in range(16):** // as there are 16 rounds

**plaintext=self.round(i,plaintext)** // sending plaintext to round

**right=plaintext[:8]**

**left=plaintext[8:]**

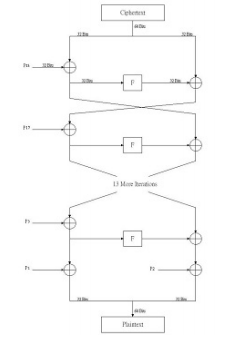
**right =self.xor(right,self.p[16])** // xor right with P 16 value

**left =self.xor(left,self.p[17])** // xor left with P 17 value

**return left+right**  // returning the encrypted text

**Decryption**

It uses the same method as of encryption only the keys are in backward order i.e from P17 in round 1 to P0 in round 16.



**CODE**

**def round(self,i,plaintext):**

**left=plaintext[:8]**

**right=plaintext[8:]**

**left =self.xor(left, self.p[i])**

**if len(left)<8:**

**left="0" +left[:]**

**fOut=self.f(left)**

**right =self.xor(fOut,right)**

**if len(right)<8:**

**right ="0" +right[:]**

**#print("round "+ str(i)+" : "+ right + left)**

**return right + left**

**def decrypt(self,plaintext):**

**for i in range(17,1,-1):** // starting the P values from backward i.e P 17 to P 0

**plaintext=self.round(i,plaintext)**

**right=plaintext[:8]**

**left=plaintext[8:]**

**right =self.xor(right,self.p[1])**

**left =self.xor(left,self.p[0])**

**return left+right**

**Result**

The above model has been implemented in python without using any cryptographic libraries and for a file of 1kb to encryption is done in 127.6ms and decryption in 129.9ms without any data loss.

**Conclusion**

The experiment is a method that shows the ciphertext recovery from cloud storage implemented in python, which is much more secure, less time consuming and lighter than DES and other conventional algorithms.

**References**

<https://ieeexplore.ieee.org/document/8378204>

<https://ieeexplore.ieee.org/document/8250724>

<https://ieeexplore.ieee.org/document/8250724>

[https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/amp](https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/amp/)

<https://www.researchgate.net/publication/324251214_Performance_Evaluation_of_Blowfish_Algorithm_on_Supercomputer_IMAN1>