EXPERIMENT NO 3

Aim : Implementation of Blowfish algorithm

Theory:

What is a Block Cipher?

A block cipher is an encryption method that applies a deterministic algorithm along with a symmetric key to encrypt a block of text, rather than encrypting one bit at a time as in stream ciphers. For example, a common block cipher, AES, encrypts 128 bit blocks with a key of predetermined length: 128, 192, or 256 bits. Block ciphers are pseudorandom permutation (PRP) families that operate on the fixed size block of bits. PRPs are functions that cannot be differentiated from completely random permutations and thus, are considered reliable, until proven unreliable.

Block cipher modes of operation have been developed to eliminate the chance of encrypting identical blocks of text the same way, the ciphertext formed from the previous encrypted block is applied to the next block. A block of bits called an initialization vector (IV) is also used by modes of operation to ensure ciphertext remain distinct even when the same plaintext message is encrypted a number of times.

Some of the various modes of operation for block ciphers include CBC (cipher block chaining), CFB (cipher feedback), CTR (counter), and GCM (Galois/Counter Mode), among others. Above is an example of CBC mode.

Where an IV is crossed with the initial plaintext block and the encryption algorithm is completed with a given key and the ciphertext is then outputted. This resultant cipher text is then used in place of the IV in subsequent plaintext blocks.

What is Blowfish algorithm?

Blowfish is a symmetric encryption algorithm developed by Bruce Schneier to replace Data Encryption Standard (DES). At the time of its development, most encryption algorithms were protected by patents, government secrecy, or company intellectual property. Schneier placed Blowfish in the public domain making it freely available for anyone to use.

Blowfish is a 16-round Feistel cipher. It's block size is 64-bit and key sizes range from 32 to 448 bits. Blowfish is a fast block cipher, except when changing keys. Each new key requires pre-processing equivalent to encrypting about 4 kilobytes of text, which is very slow compared to other block ciphers. This prevents its use in certain applications, but is not a problem in others.

In one application Blowfish's slow key changing is actually a benefit: the password-hashing method used in OpenBSD uses an algorithm derived from Blowfish that makes use of the slow key schedule; the idea is that the extra computational effort required gives protection against dictionary attacks.

Blowfish has a memory footprint of just over 4 kilobytes of RAM. This constraint is not a problem even for older desktop and laptop computers, though it does prevent use in the smallest embedded systems such as early smartcards.

Blowfish was one of the first secure block ciphers not subject to any patents and therefore freely available for anyone to use. This benefit has contributed to its popularity in cryptographic software.

Advantages and disadvantages of Blowfish Algorithm

Advantages :

- Blowfish is in the public domain, allowing it to be freely used for any purpose.

- After the key schedule has completed, Blowfish is a relatively fast block cipher due to the small number of rounds (sixteen) and the simplicity of the round operation (a few modular additions and exclusive-ors).

Disadvantages :

- The key schedule in Blowfish is rather time-consuming (equivalent to encryption of about 4 KB of data). However, this can be an advantage in some circumstances as protection against brute-force attacks.

- The small block size of Blowfish (64 bits) is more vulnerable to birthday attacks than the 128 bits used by AES.

- The creator of Blowfish, Bruce Schneier, recommends that Blowfish be abandoned in favor of Twofish, a cipher of which he was part of the development team.

Areas Of Applications :

A standard encryption algorithm must be suitable for many different applications:

1. Bulk encryption: The algorithm should be efficient in encrypting data files or a continuous data stream.
2. Random bit generation: The algorithm should be efficient in producing single random bits.
3. Packet encryption: The algorithm should be efficient in encrypting packet-sized data. (An ATM packet has a 48- byte data field.) It should implementable in an application where successive packets may be encrypted or decrypted with different keys.
4. Hashing: The algorithm should be efficient in being converted to a one-way hash function.

Algorithm:

Operations: (Blowfish encrypts 64-bit block with a variable length key)

1) **Blowfish Key Schedule (and S-box generation)**

Initial values

The Blowfish key schedule relies heavily on the Blowfish encryption algorithm described in the previous section. The key schedule uses a value, P, consisting of eighteen words which contain (in order) the first eighteen words of the binary representation of the fractional part of pi. For example, the hexadecimal representation of pi begins with 3.243F6A8885A308D313198A2E037073, therefore P1=0x243F6A88, P2=0x85A308D3, etc. This value, P, will become the round keys used in encryption.

Next, set the initial values of the S-Boxes in the same manner beginning with the 19th word of the fractional part of pi. The ordering should be that the entire first S-Box is filled in order before moving on to the next and so on.

Since P contains 18 words and the S-Boxes each contain 256 words, a total of 18 + 4*256 = 1042 pi words are used, each 32-bit in size.

Generating Round Keys and S-box

Generation of the round key is performed in rounds where each round generates two round key values. The process is as follows:

1. Initialize P and S-Boxes as described above
2. Exclusive-or P1 with the first 32 key bits, P2 with the next 32 bits and so on until all of the key has been exclusive-ored (since the key is shorter than P, parts of it will be used multiple times to cover all of P)
3. Set the initial input to zero
4. Encrypt the input using the current version of P as the round keys
5. Set the first two unreplaced values of P to the value of the ciphertext from step 4
6. Set the input to the ciphertext from step 4
7. Repeat steps 4 through 6 until all of P has been replaced
8. Use the resulting value of P as the round keys in encryption
9. Repeat steps 4 through 6, replacing values of the S-Boxes two at a time until all S-Box values have been replaced.

Since P contains 18 words and the S-Boxes each contain 256 words, there is a total of 18 + 4*256 = 1042 values to replace, which will take 521 iterations of steps 4 through 6 of the above algorithm to complete.

The algorithm for function F is:

- Given a 32-bit value X, break it in 4 bytes: a, b, c and d.

- Return ((S1[a] + S2[b]) xor S3[c]) + S4[d]

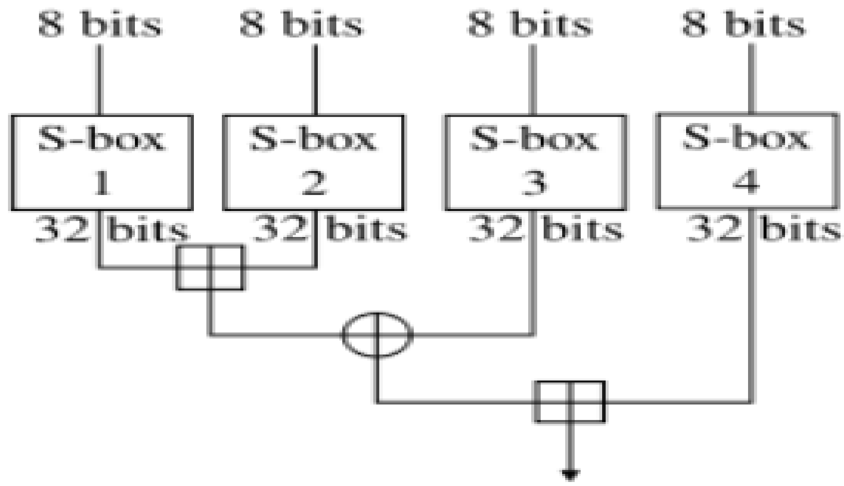

Figure 1: Function F

2) Data Encryption

The encryption algorithm thus become:

- Split the plain 64-bit value X in two 32-bit halves: Xl and Xr
- For i = 1 to 16 do
    - Xl = Xl xor P[i]
    - Xr = Xr or F(Xl)
    - Swap Xl and Xr
- Undo the last swap
- Xr = Xr xor P[17]
- Xl = Xl xor P[18]
- Recombine Xl and Xr into a ciphered 64-bit value

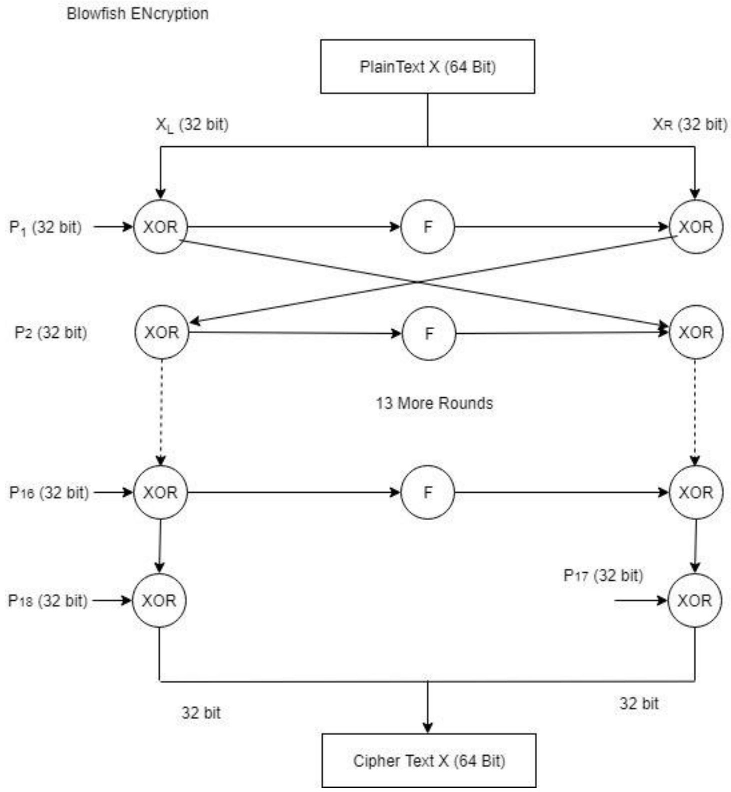

Figure 2: Blowfish Encryption

3) Data Decryption

To decode the final value back to the original one, the same method is used, except that the array P is walked backwards:

- Split the ciphered 64-bit value X in two 32-bit halves: Xl and Xr
- For i =18 to 3 step -1 do
    - Xl = Xl xor P[i]
    - Xr = Xr or F(Xl)
    - Swap Xl and Xr
- Undo the last swap
- Xr = Xr xor P[2]
- Xl = Xl xor P[1]
- Recombine Xl and Xr into a plain 64-bit value

Code:

```
#************************************************************************

import binascii

from random import randint
```

p_box=[

0x243f6a88, 0x85a308d3, 0x13198a2e, 0x03707344, 0xa4093822, 0x299f31d0,

0x082efa98, 0xec4e6c89, 0x452821e6, 0x38d01377, 0xbe5466cf, 0x34e90c6c,

0xc0ac29b7, 0xc97c50dd, 0x3f84d5b5, 0xb5470917, 0x9216d5d9, 0x8979fb1b]

s_box_1=[

0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7, 0xb8e1afed, 0x6a267e96,

0xba7c9045, 0xf12c7f99, 0x24a19947, 0xb3916cf7, 0x0801f2e2, 0x858efc16,

0x636920d8, 0x71574e69, 0xa458fea3, 0xf4933d7e, 0x0d95748f, 0x728eb658,

0x718bcd58, 0x82154aee, 0x7b54a41d, 0xc25a59b5, 0x9c30d539, 0x2af26013,

0xc5d1b023, 0x286085f0, 0xca417918, 0xb8db38ef, 0x8e79dcb0, 0x603a180e,

0x6c9e0e8b, 0xb01e8a3e, 0xd71577c1, 0xbd314b27, 0x78af2fda, 0x55605c60,

0xe65525f3, 0xaa55ab94, 0x57489862, 0x63e81440, 0x55ca396a, 0x2aab10b6,

0xb4cc5c34, 0x1141e8ce, 0xa15486af, 0x7c72e993, 0xb3ee1411, 0x636fbc2a,

0x2ba9c55d, 0x741831f6, 0xce5c3e16, 0x9b87931e, 0xafd6ba33, 0x6c24cf5c,

0x7a325381, 0x28958677, 0x3b8f4898, 0x6b4bb9af, 0xc4bfe81b, 0x66282193,

0x61d809cc, 0xfb21a991, 0x487cac60, 0x5dec8032, 0xef845d5d, 0xe98575b1,

0xdc262302, 0xeb651b88, 0x23893e81, 0xd396acc5, 0x0f6d6ff3, 0x83f44239,

0x2e0b4482, 0xa4842004, 0x69c8f04a, 0x9e1f9b5e, 0x21c66842, 0xf6e96c9a,

0x670c9c61, 0xabd388f0, 0x6a51a0d2, 0xd8542f68, 0x960fa728, 0xab5133a3,

0x6eef0b6c, 0x137a3be4, 0xba3bf050, 0x7efb2a98, 0xa1f1651d, 0x39af0176,

0x66ca593e, 0x82430e88, 0x8cee8619, 0x456f9fb4, 0x7d84a5c3, 0x3b8b5ebe,

0xe06f75d8, 0x85c12073, 0x401a449f, 0x56c16aa6, 0x4ed3aa62, 0x363f7706,

0x1bfedf72, 0x429b023d, 0x37d0d724, 0xd00a1248, 0xdb0fead3, 0x49f1c09b, 0x075372c9, 0x80991b7b, 0x25d479d8, 0xf6e8def7, 0xe3fe501a, 0xb6794c3b,

0x976ce0bd, 0x04c006ba, 0xc1a94fb6, 0x409f60c4, 0x5e5c9ec2, 0x196a2463,

0x68fb6faf, 0x3e6c53b5, 0x1339b2eb, 0x3b52ec6f, 0x6dfc511f, 0x9b30952c,

0xcc814544, 0xaf5ebd09, 0xbee3d004, 0xde334afd, 0x660f2807, 0x192e4bb3,

0xc0cba857, 0x45c8740f, 0xd20b5f39, 0xb9d3fbdb, 0x5579c0bd, 0x1a60320a,

0xd6a100c6, 0x402c7279, 0x679f25fe, 0xfb1fa3cc, 0x8ea5e9f8, 0xdb3222f8,

0x3c7516df, 0xfd616b15, 0x2f501ec8, 0xad0552ab, 0x323db5fa, 0xfd238760,

0x53317b48, 0x3e00df82, 0x9e5c57bb, 0xca6f8ca0, 0x1a87562e, 0xdf1769db,

0xd542a8f6, 0x287effc3, 0xac6732c6, 0x8c4f5573, 0x695b27b0, 0xbbca58c8,

0xe1ffa35d, 0xb8f011a0, 0x10fa3d98, 0xfd2183b8, 0x4afcb56c, 0x2dd1d35b,

0x9a53e479, 0xb6f84565, 0xd28e49bc, 0x4bfb9790, 0xe1ddf2da, 0xa4cb7e33,

0x62fb1341, 0xcee4c6e8, 0xef20cada, 0x36774c01, 0xd07e9efe, 0x2bf11fb4,

0x95dbda4d, 0xae909198, 0xeaad8e71, 0x6b93d5a0, 0xd08ed1d0, 0xafc725e0,

0x8e3c5b2f, 0x8e7594b7, 0x8ff6e2fb, 0xf2122b64, 0x8888b812, 0x900df01c,

0x4fad5ea0, 0x688fc31c, 0xd1cff191, 0xb3a8c1ad, 0x2f2f2218, 0xbe0e1777,

0xea752dfe, 0x8b021fa1, 0xe5a0cc0f, 0xb56f74e8, 0x18acf3d6, 0xce89e299,

0xb4a84fe0, 0xfd13e0b7, 0x7cc43b81, 0xd2ada8d9, 0x165fa266, 0x80957705,

0x93cc7314, 0x211a1477, 0xe6ad2065, 0x77b5fa86, 0xc75442f5, 0xfb9d35cf,

0xebcdaf0c, 0x7b3e89a0, 0xd6411bd3, 0xae1e7e49, 0x00250e2d, 0x2071b35e,

0x226800bb, 0x57b8e0af, 0x2464369b, 0xf009b91e, 0x5563911d, 0x59dfa6aa,

0x78c14389, 0xd95a537f, 0x207d5ba2, 0x02e5b9c5, 0x83260376, 0x6295cfa9,

0x11c81968, 0x4e734a41, 0xb3472dca, 0x7b14a94a, 0x1b510052, 0x9a532915,

0xd60f573f, 0xbc9bc6e4, 0x2b60a476, 0x81e67400, 0x08ba6fb5, 0x571be91f,

0xf296ec6b, 0x2a0dd915, 0xb6636521, 0xe7b9f9b6, 0xff34052e, 0xc5855664,

0x53b02d5d, 0xa99f8fa1, 0x08ba4799, 0x6e85076a]

```
s_box_2=[
    0x4b7a70e9, 0xb5b32944,
    0xdb75092e, 0xc4192623, 0xad6ea6b0, 0x49a7df7d, 0x9cee60b8, 0x8fedb266,
    0xecaa8c71, 0x699a17ff, 0x5664526c, 0xc2b19ee1, 0x193602a5, 0x75094c29,
    0xa0591340, 0xe4183a3e, 0x3f54989a, 0x5b429d65, 0x6b8fe4d6, 0x99f73fd6,
    0xa1d29c07, 0xefe830f5, 0x4d2d38e6, 0xf0255dc1, 0x4cdd2086, 0x8470eb26,
    0x6382e9c6, 0x021ecc5e, 0x09686b3f, 0x3ebaefc9, 0x3c971814, 0x6b6a70a1,
    0x687f3584, 0x52a0e286, 0xb79c5305, 0xaa500737, 0x3e07841c, 0x7fdeae5c,
    0x8e7d44ec, 0x5716f2b8, 0xb03ada37, 0xf0500c0d, 0xf01c1f04, 0x0200b3ff,
    0xae0cf51a, 0x3cb574b2, 0x25837a58, 0xdc0921bd, 0xd19113f9, 0x7ca92ff6,
    0x94324773, 0x22f54701, 0x3ae5e581, 0x37c2dadc, 0xc8b57634, 0x9af3dda7,
    0xa9446146, 0x0fd0030e, 0xecc8c73e, 0xa4751e41, 0xe238cd99, 0x3bea0e2f,
    0x3280bba1, 0x183eb331, 0x4e548b38, 0x4f6db908, 0x6f420d03, 0xf60a04bf,
    0x2cb81290, 0x24977c79, 0x5679b072, 0xbcaf89af, 0xde9a771f, 0xd9930810,
    0xb38bae12, 0xdccf3f2e, 0x5512721f, 0x2e6b7124, 0x501adde6, 0x9f84cd87,
    0x7a584718, 0x7408da17, 0xbc9f9abc, 0xe94b7d8c, 0xec7aec3a, 0xdb851dfa,
    0x63094366, 0xc464c3d2, 0xef1c1847, 0x3215d908, 0xdd433b37, 0x24c2ba16,
    0x12a14d43, 0x2a65c451, 0x50940002, 0x133ae4dd, 0x71dff89e, 0x10314e55,
    0x81ac77d6, 0x5f11199b, 0x043556f1, 0xd7a3c76b, 0x3c11183b, 0x5924a509,
    0xf28fe6ed, 0x97f1fbfa, 0x9ebabf2c, 0x1e153c6e, 0x86e34570, 0xeae96fb1,
    0x860e5e0a, 0x5a3e2ab3, 0x771fe71c, 0x4e3d06fa, 0x2965dcb9, 0x99e71d0f,
    0x803e89d6, 0x5266c825, 0x2e4cc978, 0x9c10b36a, 0xc6150eba, 0x94e2ea78,
    0xa5fc3c53, 0x1e0a2df4, 0xf2f74ea7, 0x361d2b3d, 0x1939260f, 0x19c27960,
    0x5223a708, 0xf71312b6, 0xebadfe6e, 0xeac31f66, 0xe3bc4595, 0xa67bc883,
```

0xb17f37d1, 0x018cff28, 0xc332ddef, 0xbe6c5aa5, 0x65582185, 0x68ab9802,

0xeecea50f, 0xdb2f953b, 0x2aef7dad, 0x5b6e2f84, 0x1521b628, 0x29076170,

0xecdd4775, 0x619f1510, 0x13cca830, 0xeb61bd96, 0x0334fe1e, 0xaa0363cf,

0xb5735c90, 0x4c70a239, 0xd59e9e0b, 0xcbaade14, 0xeecc86bc, 0x60622ca7,

0x9cab5cab, 0xb2f3846e, 0x648b1eaf, 0x19bdf0ca, 0xa02369b9, 0x655abb50,

0x40685a32, 0x3c2ab4b3, 0x319ee9d5, 0xc021b8f7, 0x9b540b19, 0x875fa099,

0x95f7997e, 0x623d7da8, 0xf837889a, 0x97e32d77, 0x11ed935f, 0x16681281,

0x0e358829, 0xc7e61fd6, 0x96dedfa1, 0x7858ba99, 0x57f584a5, 0x1b227263,

0x9b83c3ff, 0x1ac24696, 0xcdb30aeb, 0x532e3054, 0x8fd948e4, 0x6dbc3128,

0x58ebf2ef, 0x34c6ffea, 0xfe28ed61, 0xee7c3c73, 0x5d4a14d9, 0xe864b7e3,

0x42105d14, 0x203e13e0, 0x45eee2b6, 0xa3aaabea, 0xdb6c4f15, 0xfacb4fd0,

0xc742f442, 0xef6abbb5, 0x654f3b1d, 0x41cd2105, 0xd81e799e, 0x86854dc7,

0xe44b476a, 0x3d816250, 0xcf62a1f2, 0x5b8d2646, 0xfc8883a0, 0xc1c7b6a3,

0x7f1524c3, 0x69cb7492, 0x47848a0b, 0x5692b285, 0x095bbf00, 0xad19489d,

0x1462b174, 0x23820e00, 0x58428d2a, 0x0c55f5ea, 0x1dadf43e, 0x233f7061,

0x3372f092, 0x8d937e41, 0xd65fecf1, 0x6c223bdb, 0x7cde3759, 0xcbee7460,

0x4085f2a7, 0xce77326e, 0xa6078084, 0x19f8509e, 0xe8efd855, 0x61d99735,

0xa969a7aa, 0xc50c06c2, 0x5a04abfc, 0x800bcadc, 0x9e447a2e, 0xc3453484,

0xfdd56705, 0x0e1e9ec9, 0xdb73dbd3, 0x105588cd, 0x675fda79, 0xe3674340,

0xc5c43465, 0x713e38d8, 0x3d28f89e, 0xf16dff20, 0x153e21e7, 0x8fb03d4a,

0xe6e39f2b, 0xdb83adf7]

s_box_3=[

0xe93d5a68, 0x948140f7, 0xf64c261c, 0x94692934,

0x411520f7, 0x7602d4f7, 0xbcf46b2e, 0xd4a20068, 0xd4082471, 0x3320f46a,

0x43b7d4b7, 0x500061af, 0x1e39f62e, 0x97244546, 0x14214f74, 0xbf8b8840,

0x4d95fc1d, 0x96b591af, 0x70f4ddd3, 0x66a02f45, 0xbfbc09ec, 0x03bd9785,

0x7fac6dd0, 0x31cb8504, 0x96eb27b3, 0x55fd3941, 0xda2547e6, 0xabca0a9a,

0x28507825, 0x530429f4, 0x0a2c86da, 0xe9b66dfb, 0x68dc1462, 0xd7486900,

0x680ec0a4, 0x27a18dee, 0x4f3ffea2, 0xe887ad8c, 0xb58ce006, 0x7af4d6b6,

0xaace1e7c, 0xd3375fec, 0xce78a399, 0x406b2a42, 0x20fe9e35, 0xd9f385b9,

0xee39d7ab, 0x3b124e8b, 0x1dc9faf7, 0x4b6d1856, 0x26a36631, 0xeae397b2,

0x3a6efa74, 0xdd5b4332, 0x6841e7f7, 0xca7820fb, 0xfb0af54e, 0xd8feb397,

0x454056ac, 0xba489527, 0x55533a3a, 0x20838d87, 0xfe6ba9b7, 0xd096954b,

0x55a867bc, 0xa1159a58, 0xcca92963, 0x99e1db33, 0xa62a4a56, 0x3f3125f9,

0x5ef47e1c, 0x9029317c, 0xfdf8e802, 0x04272f70, 0x80bb155c, 0x05282ce3,

0x95c11548, 0xe4c66d22, 0x48c1133f, 0xc70f86dc, 0x07f9c9ee, 0x41041f0f,

0x404779a4, 0x5d886e17, 0x325f51eb, 0xd59bc0d1, 0xf2bcc18f, 0x41113564,

0x257b7834, 0x602a9c60, 0xdff8e8a3, 0x1f636c1b, 0x0e12b4c2, 0x02e1329e,

0xaf664fd1, 0xcad18115, 0x6b2395e0, 0x333e92e1, 0x3b240b62, 0xeebeb922,

0x85b2a20e, 0xe6ba0d99, 0xde720c8c, 0x2da2f728, 0xd0127845, 0x95b794fd,

0x647d0862, 0xe7ccf5f0, 0x5449a36f, 0x877d48fa, 0xc39dfd27, 0xf33e8d1e,

0x0a476341, 0x992eff74, 0x3a6f6eab, 0xf4f8fd37, 0xa812dc60, 0xa1ebddf8,

0x991be14c, 0xdb6e6b0d, 0xc67b5510, 0x6d672c37, 0x2765d43b, 0xdcd0e804,

0xf1290dc7, 0xcc00ffa3, 0xb5390f92, 0x690fed0b, 0x667b9ffb, 0xcedb7d9c,

0xa091cf0b, 0xd9155ea3, 0xbb132f88, 0x515bad24, 0x7b9479bf, 0x763bd6eb,

0x37392eb3, 0xcc115979, 0x8026e297, 0xf42e312d, 0x6842ada7, 0xc66a2b3b,

0x12754ccc, 0x782ef11c, 0x6a124237, 0xb79251e7, 0x06a1bbe6, 0x4bfb6350,

```
    0x1a6b1018, 0x11caedfa, 0x3d25bdd8, 0xe2e1c3c9, 0x44421659, 0x0a121386,
    0xd90cec6e, 0xd5abea2a, 0x64af674e, 0xda86a85f, 0xbebfe988, 0x64e4c3fe,
    0x9dbc8057, 0xf0f7c086, 0x60787bf8, 0x6003604d, 0xd1fd8346, 0xf6381fb0,
    0x7745ae04, 0xd736fccc, 0x83426b33, 0xf01eab71, 0xb0804187, 0x3c005e5f,
    0x77a057be, 0xbde8ae24, 0x55464299, 0xbf582e61, 0x4e58f48f, 0xf2ddfda2,
    0xf474ef38, 0x8789bdc2, 0x5366f9c3, 0xc8b38e74, 0xb475f255, 0x46fcd9b9,
    0x7aeb2661, 0x8b1ddf84, 0x846a0e79, 0x915f95e2, 0x466e598e, 0x20b45770,
    0x8cd55591, 0xc902de4c, 0xb90bace1, 0xbb8205d0, 0x11a86248, 0x7574a99e,
    0xb77f19b6, 0xe0a9dc09, 0x662d09a1, 0xc4324633, 0xe85a1f02, 0x09f0be8c,
    0x4a99a025, 0x1d6efe10, 0x1ab93d1d, 0x0ba5a4df, 0xa186f20f, 0x2868f169,
    0xdcb7da83, 0x573906fe, 0xa1e2ce9b, 0x4fcd7f52, 0x50115e01, 0xa70683fa,
    0xa002b5c4, 0x0de6d027, 0x9af88c27, 0x773f8641, 0xc3604c06, 0x61a806b5,
    0xf0177a28, 0xc0f586e0, 0x006058aa, 0x30dc7d62, 0x11e69ed7, 0x2338ea63,
    0x53c2dd94, 0xc2c21634, 0xbbcbee56, 0x90bcb6de, 0xebfc7da1, 0xce591d76,
    0x6f05e409, 0x4b7c0188, 0x39720a3d, 0x7c927c24, 0x86e3725f, 0x724d9db9,
    0x1ac15bb4, 0xd39eb8fc, 0xed545578, 0x08fca5b5, 0xd83d7cd3, 0x4dad0fc4,
    0x1e50ef5e, 0xb161e6f8, 0xa28514d9, 0x6c51133c, 0x6fd5c7e7, 0x56e14ec4,
    0x362abfce, 0xddc6c837, 0xd79a3234, 0x92638212, 0x670efa8e, 0x406000e0]

s_box_4=[
    0x3a39ce37, 0xd3faf5cf, 0xabc27737, 0x5ac52d1b, 0x5cb0679e, 0x4fa33742,
    0xd3822740, 0x99bc9bbe, 0xd5118e9d, 0xbf0f7315, 0xd62d1c7e, 0xc700c47b,
    0xb78c1b6b, 0x21a19045, 0xb26eb1be, 0x6a366eb4, 0x5748ab2f, 0xbc946e79,
    0xc6a376d2, 0x6549c2c8, 0x530ff8ee, 0x468dde7d, 0xd5730a1d, 0x4cd04dc6,
    0x2939bbdb, 0xa9ba4650, 0xac9526e8, 0xbe5ee304, 0xa1fad5f0, 0x6a2d519a,
```

0x63ef8ce2, 0x9a86ee22, 0xc089c2b8, 0x43242ef6, 0xa51e03aa, 0x9cf2d0a4,

0x83c061ba, 0x9be96a4d, 0x8fe51550, 0xba645bd6, 0x2826a2f9, 0xa73a3ae1,

0x4ba99586, 0xef5562e9, 0xc72fefd3, 0xf752f7da, 0x3f046f69, 0x77fa0a59,

0x80e4a915, 0x87b08601, 0x9b09e6ad, 0x3b3ee593, 0xe990fd5a, 0x9e34d797,

0x2cf0b7d9, 0x022b8b51, 0x96d5ac3a, 0x017da67d, 0xd1cf3ed6, 0x7c7d2d28,

0x1f9f25cf, 0xadf2b89b, 0x5ad6b472, 0x5a88f54c, 0xe029ac71, 0xe019a5e6,

0x47b0acfd, 0xed93fa9b, 0xe8d3c48d, 0x283b57cc, 0xf8d56629, 0x79132e28,

0x785f0191, 0xed756055, 0xf7960e44, 0xe3d35e8c, 0x15056dd4, 0x88f46dba,

0x03a16125, 0x0564f0bd, 0xc3eb9e15, 0x3c9057a2, 0x97271aec, 0xa93a072a,

0x1b3f6d9b, 0x1e6321f5, 0xf59c66fb, 0x26dcf319, 0x7533d928, 0xb155fdf5,

0x03563482, 0x8aba3cbb, 0x28517711, 0xc20ad9f8, 0xabcc5167, 0xccad925f,

0x4de81751, 0x3830dc8e, 0x379d5862, 0x9320f991, 0xea7a90c2, 0xfb3e7bce,

0x5121ce64, 0x774fbe32, 0xa8b6e37e, 0xc3293d46, 0x48de5369, 0x6413e680,

0xa2ae0810, 0xdd6db224, 0x69852dfd, 0x09072166, 0xb39a460a, 0x6445c0dd,

0x586cdecf, 0x1c20c8ae, 0x5bbef7dd, 0x1b588d40, 0xccd2017f, 0x6bb4e3bb,

0xdda26a7e, 0x3a59ff45, 0x3e350a44, 0xbcb4cdd5, 0x72eacea8, 0xfa6484bb,

0x8d6612ae, 0xbf3c6f47, 0xd29be463, 0x542f5d9e, 0xaec2771b, 0xf64e6370,

0x740e0d8d, 0xe75b1357, 0xf8721671, 0xaf537d5d, 0x4040cb08, 0x4eb4e2cc,

0x34d2466a, 0x0115af84, 0xe1b00428, 0x95983a1d, 0x06b89fb4, 0xce6ea048,

0x6f3f3b82, 0x3520ab82, 0x011a1d4b, 0x277227f8, 0x611560b1, 0xe7933fdc,

0xbb3a792b, 0x344525bd, 0xa08839e1, 0x51ce794b, 0x2f32c9b7, 0xa01fbac9,

0xe01cc87e, 0xbcc7d1f6, 0xcf0111c3, 0xa1e8aac7, 0x1a908749, 0xd44fbd9a,

0xd0dadecb, 0xd50ada38, 0x0339c32a, 0xc6913667, 0x8df9317c, 0xe0b12b4f,

0xf79e59b7, 0x43f5bb3a, 0xf2d519ff, 0x27d9459c, 0xbf97222c, 0x15e6fc2a,

0x0f91fc71, 0x9b941525, 0xfae59361, 0xceb69ceb, 0xc2a86459, 0x12baa8d1,

```
    0xb6c1075e, 0xe3056a0c, 0x10d25065, 0xcb03a442, 0xe0ec6e0e, 0x1698db3b,
    0x4c98a0be, 0x3278e964, 0x9f1f9532, 0xe0d392df, 0xd3a0342b, 0x8971f21e,
    0x1b0a7441, 0x4ba3348c, 0xc5be7120, 0xc37632d8, 0xdf359f8d, 0x9b992f2e,
    0xe60b6f47, 0x0fe3f11d, 0xe54cda54, 0x1edad891, 0xce6279cf, 0xcd3e7e6f,
    0x1618b166, 0xfd2c1d05, 0x848fd2c5, 0xf6fb2299, 0xf523f357, 0xa6327623,
    0x93a83531, 0x56cccd02, 0xacf08162, 0x5a75ebb5, 0x6e163697, 0x88d273cc,
    0xde966292, 0x81b949d0, 0x4c50901b, 0x71c65614, 0xe6c6c7bd, 0x327a140a,
    0x45e1d006, 0xc3f27b9a, 0xc9aa53fd, 0x62a80f00, 0xbb25bfe2, 0x35bdd2f6,
    0x71126905, 0xb2040222, 0xb6cbcf7c, 0xcd769c2b, 0x53113ec0, 0x1640e3d3,
    0x38abbd60, 0x2547adf0, 0xba38209c, 0xf746ce76, 0x77afa1c5, 0x20756060,
    0x85cbfe4e, 0x8ae88dd8, 0x7aaaf9b0, 0x4cf9aa7e, 0x1948c25c, 0x02fb8a8c,
    0x01c36ae4, 0xd6ebe1f9, 0x90d4f869, 0xa65cdea0, 0x3f09252d, 0xc208e69f,
    0xb74e6132, 0xce77e25b, 0x578fdfe3, 0x3ac372e6]

key_temp=[ 0x58ebf2ef, 0x34c6ffea, 0xfe28ed61, 0xee7c3c73, 0x5d4a14d9, 0xe864b7e3,
    0x42105d14, 0x203e13e0, 0x45eee2b6, 0xa3aaabea, 0xdb6c4f15, 0xfacb4fd0,
    0xc742f442, 0xef6abbb5 ]
key_length=randint(1,14)
k=0
key=[]
while k<key_length:
    key.append(s_box_3[randint(0,255)])
    k+=1
print(key)
```

```
def blowfish_algo(pair):
    i=0
    while i<=15:
        pair[0]=pair[0]^p_box[i]
        pair[1]=function_f(pair[0])^pair[1]
        temp=pair[0]
        pair[0]=pair[1]
        pair[1]=temp
        i+=1
    temp = pair[0]
    pair[0] = pair[1]
    pair[1] = temp
    pair[1] = pair[1]^p_box[16]
    pair[0] = pair[0]^p_box[17]
    return pair


def blowfish_algo_decrypt(pair):
    i=17
    while i>=2:
        pair[0]=pair[0]^p_box[i]
        pair[1]=function_f(pair[0])^pair[1]
        temp=pair[0]
        pair[0]=pair[1]
        pair[1]=temp
```

```
        i-=1
    temp = pair[0]
    pair[0] = pair[1]
    pair[1] = temp
    pair[1] = pair[1]^p_box[1]
    pair[0] = pair[0]^p_box[0]
    return pair


def function_f(temp):
    binary='{:032b}'.format(temp)
    a=int(binary[0:8], 2)
    b=int(binary[8:16], 2)
    c=int(binary[16:24], 2)
    d=int(binary[24:32], 2)
    return (( ( (s_box_1[a]+s_box_2[b])%(2**32))^s_box_3[c])+s_box_4[d])%(2**32)



p_index=0
key_index=0
while p_index<len(p_box):
    p_box[p_index]=p_box[p_index]^key[key_index]
    p_index+=1
    key_index=(key_index+1)%len(key)
```

```
pair=[0x00000000, 0x00000000]
i=0
temp=[]
while i<len(p_box):
    pair=blowfish_algo(pair)
    p_box[i]=pair[0]
    p_box[i+1]=pair[1]
    i+=2
i=0
while i<len(s_box_1):
    pair=blowfish_algo(pair)
    s_box_1[i]=pair[0]
    s_box_1[i+1]=pair[1]
    i+=2
i=0
while i<len(s_box_2):
    pair=blowfish_algo(pair)
    s_box_2[i]=pair[0]
    s_box_2[i+1]=pair[1]
    i+=2
i=0
while i<len(s_box_3):
    pair=blowfish_algo(pair)
    s_box_3[i]=pair[0]
    s_box_3[i+1]=pair[1]
```

```
    i+=2
i=0
while i<len(s_box_4):
    pair=blowfish_algo(pair)
    s_box_4[i]=pair[0]
    s_box_4[i+1]=pair[1]
    i+=2

text=[0x3278e964, 0x9f1f9532]
plain_text=input("Enter plain Text : ")
textnum=""
for c in plain_text:
    if c in [' ', '{', '}', '|']:
        plain_text.replace(c, '')
        continue
    textnum = textnum + str(ord(c) - 23)
text_num = int(textnum)
print(text_num)
bin_text="{0:b}".format(text_num)
print(bin_text)
print(len(bin_text))
if(len(bin_text)%64!=0):
    if len(bin_text)%64<64:
        rem=64-(len(bin_text)%64)
    else:
```

```
        rem=len(bin_text)%64
    for i in range(0,rem):
        bin_text="0"+bin_text
print("Length of binary text : ",bin_text)
bin_text_array=[]
s=0
e=64
while s < len(bin_text):
    bin_text_array.append(bin_text[s:e])
    s+=64
    e+=64
print(bin_text_array)
bin_text_encrypt=[]
bin_text_decrypt=[]
i=0
print("Text sent : \n [ ",end="")
while i<len(bin_text_array):
    text=[0x00000000, 0x00000000]
    text[0]=int(bin_text_array[i][0:32], 2)
    text[1]=int(bin_text_array[i][32:64],2)
    print(text[0],",",text[1],",",end="")
    encrypt=blowfish_algo(text)
    bin_text_encrypt+=encrypt
    decrypt=blowfish_algo_decrypt(encrypt)
    bin_text_decrypt+=decrypt
```

```
    i+=1
print(" ] ")
print("Encrypted text : ",bin_text_encrypt)
print("Decrypted text : ",bin_text_decrypt)
bin_array=[]
for temp in bin_text_decrypt:
    temp1='{:032b}'.format(temp)
    bin_array+=temp1
text_in_int=int(''.join(bin_array),2)
print(text_in_int)
str_text=str(text_in_int)
print("Decrypted text : ")
i=0
while i <= (len(str_text) - 2):
    print(chr(int(str(str_text[i]) + str(str_text[i + 1])) + 23), end="")
    i += 2
print("\n")
```

Output:

```
# "C:\Users\Austrin Dabre\Desktop\Pycharm\blowfish\venv\Scripts\python.exe" "C:/Users/Austrin Dabre/Desktop/Pycharm/blowfish/blowfish2.py"

# [677966185, 2489919796, 1779581495, 2006996926, 3641502830, 2006996926, 628848692, 3713745714, 2702547544, 2438960610, 2455994898, 3329971472, 1136121015, 1638401717]

# Enter plain Text : Hi@this@is@Rstar

# 49824193818292418292415992937491
```

#
1001110100110111101010011100101001100111101001001011110010100001101111000000011101110011111011110000010011

106

Length of binary text :
00000000000000000000001001110100110111101010011100101001100111101001001011110010100001101111000000011101110011111011110000010011

['0000000000000000000000100111010011011110101001110010100110011110',
'1001001011110010100001101111000000011101110011111011110000010011']

Text sent :

[628 , 3735497118 ,2465367792 , 500153363 ,]

Encrypted text : [2380104819, 594881994, 1841690355, 3033669892]

Decrypted text : [628, 3735497118, 2465367792, 500153363]

4982419381829241829241599293749

Decrypted text :

Hi@this@is@Rstar

Process finished with exit code 0

Conclusion: From this experiment I understood the concept of Blowfish algorithm .Also it has been observed that Blowfish is Fast, Compact, Simple and Secure. We can use blowfish algorithm for encryption and decryption in networking application.