Blowfish Final Report

by:
Wesley Wigham
Stephen Yingling
Chad Zawistowski

1. Algorithm Description

Blowfish is a block cipher developed by Bruce Schneier in 1993. It has a relatively expensive S-box initialization procedure, in which the S-boxes depend on the key. This function only needs to be run once for a given key (to initialize the cipher), but makes Blowfish resistant to brute-force attacks due to the expensive execution cost.

The subkey array P (18 entries) and the 4 S-boxes (256 entries each) are initialized with the hexadecimal digits of pi. When the key is set, each element of the P array is XORed with 32 bits of the key until all elements in P have been XORed with the key. The P[0] is XORed with the leftmost 32 bits of the key, P[1] with the next 32-bits and so on, cycling through the key.

Then the all 0 bitstring is encrypted using the current state and the leftmost 32-bits of the result stored in P[0] and the rightmost 32-bits in P[1]. The result is then encrypted and split into P[2] and P[3]. For every 2 elements in P, the previous result is encrypted, split in half, and assigned to P[i] and P[i+1]. After reaching the end of the P array, the same process is carried out on the S-boxes in ascending order with the result of the last encryption used for the P arrays as the input for the encryption that produces the new values for S[1][0] and S[1][1].

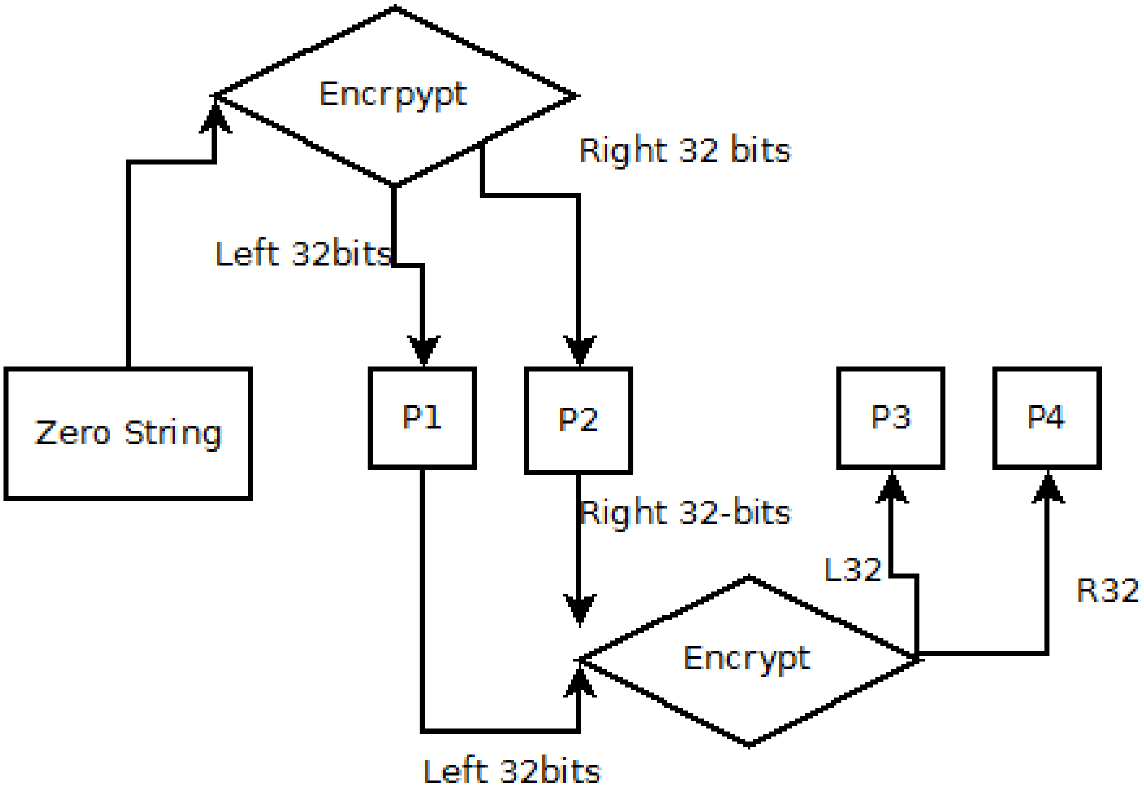

Fig 1. A diagram showing the calculation of the first 4 elements in the P array.

The encryption function is based on a feistel network with 16 rounds. The value to be encrypted is first divided into two 32-bit halves. In each round, the left half is first XORed with the round key (the nth element in the P array where n = the round number). Then the right half is XORed with F(left). Finally, the left and right halves are swapped for the next round.

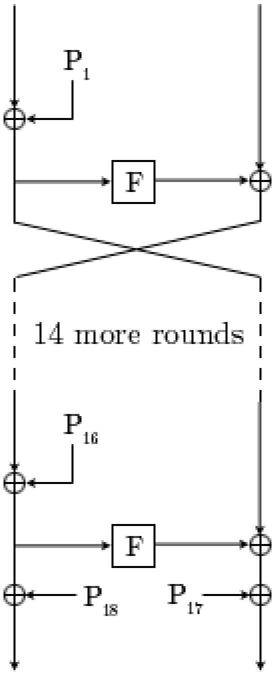

Fig 2. The feistel structure of Blowfish

After the 16th round, the halves are swapped again (effectively unswapping them), the 17th round key is XORed with the right half, and the 18th round key is XORed with the left half. The halves are then recombined as a 64-bit value and returned.

The function F() applied to the left half takes the 32-bit value and divides it into 4 bytes (a, b, c, and d where a is the most significant byte and d the least.) Each byte is then used as the input into one of the S-boxes to produce a 32-bit word. Byte a is sent through S1 and then added mod 32 to the result from sending b through S2. That result is then XORed with the result from sending c through S3. That result is then added mod 32 to the result from sending d through S4. That result is then returned.

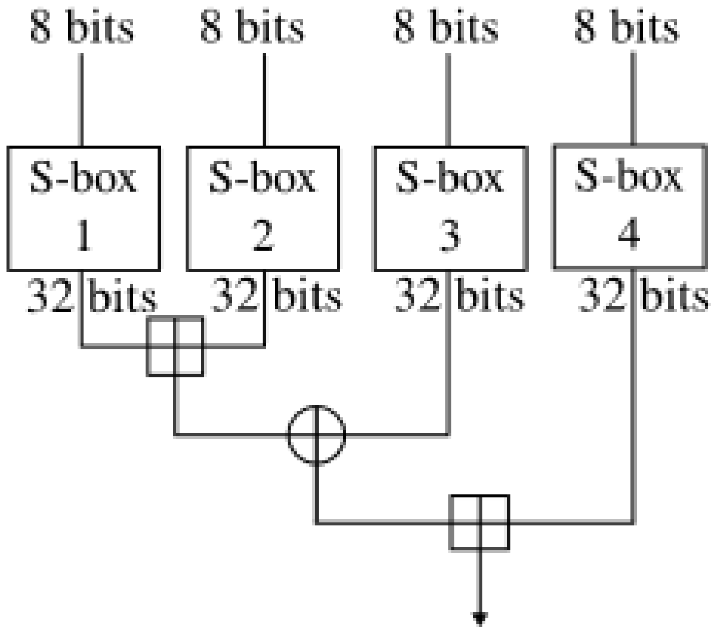

Fig 3. The function F()

2. Example inputs and outputs

key bytes	clear bytes	cipher bytes
0000000000000000	0000000000000000	4EF997456198DD78
FFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFF	51866FD5B85ECB8A
3000000000000000	1000000000000001	7D856F9A613063F2
1111111111111111	1111111111111111	2466DD878B963C9D
0123456789ABCDEF	1111111111111111	61F9C3802281B096
1111111111111111	0123456789ABCDEF	7D0CC630AFDA1EC7
0000000000000000	0000000000000000	4EF997456198DD78
FEDCBA9876543210	0123456789ABCDEF	0ACEAB0FC6A0A28D
7CA110454A1A6E57	01A1D6D039776742	59C68245EB05282B
0131D9619DC1376E	5CD54CA83DEF57DA	B1B8CC0B250F09A0
07A1133E4A0B2686	0248D43806F67172	1730E5778BEA1DA4
3849674C2602319E	51454B582DDF440A	A25E7856CF2651EB
04B915BA43FEB5B6	42FD443059577FA2	353882B109CE8F1A
0113B970FD34F2CE	059B5E0851CF143A	48F4D0884C379918
0170F175468FB5E6	0756D8E0774761D2	432193B78951FC98
43297FAD38E373FE	762514B829BF486A	13F04154D69D1AE5
07A7137045DA2A16	3BDD119049372802	2EEDDA93FFD39C79
04689104C2FD3B2F	26955F6835AF609A	D887E0393C2DA6E3
37D06BB516CB7546	164D5E404F275232	5F99D04F5B163969
1F08260D1AC2465E	6B056E18759F5CCA	4A057A3B24D3977B
584023641ABA6176	004BD6EF09176062	452031C1E4FADA8E
025816164629B007	480D39006EE762F2	7555AE39F59B87BD

49793EBC79B3258F	437540C8698F3CFA	53C55F9CB49FC019
4FB05E1515AB73A7	072D43A077075292	7A8E7BFA937E89A3
49E95D6D4CA229BF	02FE55778117F12A	CF9C5D7A4986ADB5
018310DC409B26D6	1D9D5C5018F728C2	D1ABB290658BC778
1C587F1C13924FEF	305532286D6F295A	55CB3774D13EF201
0101010101010101	0123456789ABCDEF	FA34EC4847B268B2
1F1F1F1F0E0E0E0E	0123456789ABCDEF	A790795108EA3CAE
E0FEE0FEF1FEF1FE	0123456789ABCDEF	C39E072D9FAC631D
0000000000000000	FFFFFFFFFFFFFFFF	014933E0CDAFF6E4
FFFFFFFFFFFFFFFF	0000000000000000	F21E9A77B71C49BC
0123456789ABCDEF	0000000000000000	245946885754369A
FEDCBA9876543210	FFFFFFFFFFFFFFFF	6B5C5A9C5D9E0A5A

3. Original Program Source:

```
from struct import unpack,pack

class Blowfish():

      def __init__(self,key):
      self.key = key
      #S Box is 4 rows and 256 columns (32 bit elements)
      #Initialization vectors provided by Bruce Schneier
      #https://www.schneier.com/code/constants.txt
      self.S = [
            [
                  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
            ],
            [
                  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
        ],
        [
            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
      ],
      [
            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
      ]
]
#P-array is 18 32 bit elements
self.P = [
      0x243f6a88, 0x85a308d3, 0x13198a2e, 0x03707344, 0xa4093822,
      0x299f31d0, 0x082efa98, 0xec4e6c89, 0x452821e6, 0x38d01377,
      0xbe5466cf, 0x34e90c6c, 0xc0ac29b7, 0xc97c50dd, 0x3f84d5b5,
      0xb5470917, 0x9216d5d9, 0x8979fb1b
]
self.generate_s_box()

@staticmethod
def blockSize():
"""
Returns the cipher's block size in bytes
"""
      return 8 #8 bytes = 64 bits

@staticmethod
def keySize():
"""
Returns the cipher's key size in bytes
"""
#32 bits up to 448 bits
      return 8 #64 bits!
```

```
def setKey(self, key):
"""
Sets the cipher's key
"""
      self.__init__(key)

def encrypt(self, plain):
"""
Given a plaintext block, produces a ciphertext
"""
#encrypt the block
      eblock = self.encrypt_block(plain)

#copy the encrypted block into the referenced byte array
      for i in range(8):
            plain[i] = eblock[i]

def decrypt(self, cipher):
"""
Given a ciphertext block, produces a plaintext
"""
#decrypt the block
      cblock = self.decrypt_block(cipher)

#copy the decrypted block into the referenced byte array
      for i in range(8):
            cipher[i] = cblock[i]

def generate_s_box(self):
"""
Uses the key to generate initial state s-boxes
"""


#XOR key bits into the Subkey array P
      key_len = len(self.key)
      cur_pos = 0
      for i in range(len(self.P)):
            if cur_pos+4 >= key_len:
                  next_pos = (cur_pos+4)%key_len
                  wrapped = self.key[cur_pos:]
                  wrapped.extend(self.key[:next_pos])
                  self.P[i] ^= unpack('>I',wrapped)[0]
```

```
                        cur_pos = next_pos
                  else:
                        self.P[i] ^=
unpack('>I',self.key[cur_pos:cur_pos+4])[0]
                        cur_pos += 4

      #Encrypt the all-0 string with the algorithm

            all_zero = bytearray.fromhex('00 00 00 00 00 00 00 00')

      #Assign each pair of P elements to the output
      #of the encryption of the previous two blocks
      #concatenated
            for i in range(0,len(self.P),2):
                  all_zero = self.encrypt_block(all_zero)
                  self.P[i] = unpack('>I',all_zero[0:4])[0]
                  self.P[i+1] = unpack('>I',all_zero[4:8])[0]

      #Assign each pair of S box elements to the output of
      #the encrypted values of the previous two combined
            for i in range(len(self.S)):
                  for j in range(0,len(self.S[i]),2):
                        all_zero = self.encrypt_block(all_zero)
                        self.S[i][j] = unpack('>I',all_zero[0:4])[0]
                        self.S[i][j+1] = unpack('>I',all_zero[4:8])[0]


      def feistel(self, num):
      """
      Passes a number through the feistel function
      """
      # ((S1,a + S2,b mod 2^32) XOR S3,c) + S4,d mod 2^32
      # First, divide num into 4 quarters, a, b, c, and d
            parts = pack('>I', num)
            a,b,c,d = parts[0],parts[1],parts[2],parts[3]
            return (((self.S[0][a] + self.S[1][b] % 2**32) ^ self.S[2][c])
                  + self.S[3][d]) % 2**32

      def encrypt_block(self, block):
      """
      Applies the algorithm to a block
      """
      #Make the byte array into 2 32-bit numbers
      left = unpack('>I',block[0:4])[0]
      right = unpack('>I',block[4:8])[0]

#Apply 16 rounds
      for i in range(0,16):
            left ^= self.P[i]  #XOR subkey with left
            right = self.feistel(left) ^ right #XOR right with F(left)
            left, right = right, left #swap

#unswap
      left, right = right, left

#final subkey XORing
      right ^= self.P[16]
      left ^= self.P[17]

#recreate the byte array
      ret = bytearray(pack('>I', left))
      ret.extend(pack('>I', right))
      return ret

def decrypt_block(self, block):
"""
Un-Applies the algorithm to a block
"""

#Make the byte array into 2 32-bit numbers
      left = unpack('>I',block[0:4])[0]
      right = unpack('>I',block[4:8])[0]

#Apply 16 rounds using the keys in reverse order
      for i in range(17,1,-1):
            left ^= self.P[i]  #XOR subkey with left
            right = self.feistel(left) ^ right #XOR right with F(left)
            left, right = right, left #swap

#unswap
      left, right = right, left

#XOR the final two keys
      right ^= self.P[1]
      left ^= self.P[0]
```

```
#recreate the byte array
      ret = bytearray(pack('>I', left))
      ret.extend(pack('>I', right))
return ret
```

4. Original Running Time Measurements

```
       102520883 function calls in 196.252 seconds

  Ordered by: standard name

  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       1      0.000  0.000  196.252  196.252 <string>:1(<module>)
 2500000      1.505  0.000  1.505  0.000 blowfish.py:285(blockSize)
       1      0.000  0.000  0.000  0.000 blowfish.py:292(keySize)
 2500000      6.582  0.000  183.634        0.000 blowfish.py:306(encrypt)
       1      0.002  0.002  0.042  0.042 blowfish.py:322(generate_s_box)
40008336   84.384     0.000  111.394        0.000 blowfish.py:357(feistel)
 2500521   56.802     0.000  177.092        0.000 blowfish.py:368(encrypt_block)
       1      0.000  0.000  0.042  0.042 blowfish.py:5(__init__)
       1      2.362  2.362  196.252  196.252 timetrial.py:13(test_ntimes)
 2500000      8.709  0.000  193.849        0.000 timetrial.py:19(test_once)
       1      0.000  0.000  196.252  196.252 {built-in method exec}
       1      0.000  0.000  0.000  0.000 {built-in method fromhex}
       8      0.000  0.000  0.000  0.000 {built-in method len}
45009378   30.400     0.000   30.400        0.000 {built-in method pack}
 5002102      3.761  0.000  3.761  0.000 {built-in method unpack}
       1      0.000  0.000  0.000  0.000 {method 'disable' of '_lsprof.Profiler' objects}
 2500530      1.745  0.000  1.745  0.000 {method 'extend' of 'bytearray' objects}

Done.

       102520883 function calls in 199.335 seconds

  Ordered by: standard name

  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       1      0.000  0.000  199.335  199.335 <string>:1(<module>)
 2500000      1.585  0.000  1.585  0.000 blowfish.py:285(blockSize)
       1      0.000  0.000  0.000  0.000 blowfish.py:292(keySize)
 2500000      6.605  0.000  186.475        0.000 blowfish.py:306(encrypt)
       1      0.002  0.002  0.038  0.038 blowfish.py:322(generate_s_box)
40008336   82.367     0.000  110.098        0.000 blowfish.py:357(feistel)
 2500521   60.380     0.000  179.906        0.000 blowfish.py:368(encrypt_block)
       1      0.000  0.000  0.038  0.038 blowfish.py:5(__init__)
       1      2.416  2.416  199.335  199.335 timetrial.py:13(test_ntimes)
 2500000      8.821  0.000  196.881        0.000 timetrial.py:19(test_once)
```

```
       1      0.000  0.000  199.335  199.335 {built-in method exec}
       1      0.000  0.000  0.000  0.000 {built-in method fromhex}
       8      0.000  0.000  0.000  0.000 {built-in method len}
45009378   31.308     0.000   31.308        0.000 {built-in method pack}
 5002102      3.950  0.000  3.950  0.000 {built-in method unpack}
       1      0.000  0.000  0.000  0.000 {method 'disable' of '_lsprof.Profiler' objects}
 2500530      1.901  0.000  1.901  0.000 {method 'extend' of 'bytearray' objects}

Done.

       102520883 function calls in 196.251 seconds

  Ordered by: standard name

  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       1      0.000  0.000  196.251  196.251 <string>:1(<module>)
 2500000      1.537  0.000  1.537  0.000 blowfish.py:285(blockSize)
       1      0.000  0.000  0.000  0.000 blowfish.py:292(keySize)
 2500000      6.530  0.000  183.828        0.000 blowfish.py:306(encrypt)
       1      0.002  0.002  0.044  0.044 blowfish.py:322(generate_s_box)
40008336   81.248     0.000  109.538        0.000 blowfish.py:357(feistel)
 2500521   58.256     0.000  177.340        0.000 blowfish.py:368(encrypt_block)
       1      0.000  0.000  0.044  0.044 blowfish.py:5(__init__)
       1      2.236  2.236  196.251  196.251 timetrial.py:13(test_ntimes)
 2500000      8.606  0.000  193.971        0.000 timetrial.py:19(test_once)
       1      0.000  0.000  196.251  196.251 {built-in method exec}
       1      0.000  0.000  0.000  0.000 {built-in method fromhex}
       8      0.000  0.000  0.000  0.000 {built-in method len}
45009378   31.983     0.000   31.983        0.000 {built-in method pack}
 5002102      3.966  0.000  3.966  0.000 {built-in method unpack}
       1      0.000  0.000  0.000  0.000 {method 'disable' of '_lsprof.Profiler' objects}
 2500530      1.888  0.000  1.888  0.000 {method 'extend' of 'bytearray' objects}


Done.
```

5. Analysis of Original Measurements

Most of the time is spent in the feistel function, which was called approximately 20 billion times throughout the course of the time trial. The second greatest amount of time was spent in Python’s struct.pack function, which is a flexible way to convert numbers to bytestrings (and vice-versa), but is quite inefficient when the size and format of the number is known ahead of time. Generating S-boxes is one of the slowest functions, but did not significantly affect the benchmark as the function is only called once, when the key is assigned.

6. Optimized Source Code

```
from struct import unpack,pack

class Blowfish():

      def __init__(self,key):
      self.key = key
      #S Box is 4 rows and 256 columns (32 bit elements)
      #Initialization vectors provided by Bruce Schneier
      #https://www.schneier.com/code/constants.txt
      self.S = [
            [
                  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
            ],
            [
                  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
    ],
    [
        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
    ],
    [
        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
      ]
]
#P-array is 18 32 bit elements
self.P = [
      0x243f6a88, 0x85a308d3, 0x13198a2e, 0x03707344, 0xa4093822,
      0x299f31d0, 0x082efa98, 0xec4e6c89, 0x452821e6, 0x38d01377,
      0xbe5466cf, 0x34e90c6c, 0xc0ac29b7, 0xc97c50dd, 0x3f84d5b5,
      0xb5470917, 0x9216d5d9, 0x8979fb1b
]
self.generate_s_box()

@staticmethod
def blockSize():
"""
Returns the cipher's block size in bytes
"""
return 8 #64 bits!

@staticmethod
def keySize():
"""
Returns the cipher's key size in bytes
"""
#32 bits up to 448 bits
return 8 #64 bits!
```

```
def setKey(self, key):
"""
Sets the cipher's key
"""
#reset the arrays
self.__init__(key)

def encrypt(self, plain):
"""
Given a plaintext block, produces a ciphertext
"""
#Encrypt a block
eblock = self.encrypt_block(plain)
#Add the bytes back to the referenced byte array
for i in range(8):
      plain[i] = eblock[i]

def decrypt(self, cipher):
"""
Given a ciphertext block, produces a plaintext
"""
#Encrypt a block
cblock = self.decrypt_block(cipher)
#Add the bytes back to the referenced byte array
for i in range(8):
      cipher[i] = cblock[i]

def generate_s_box(self):
"""
Uses the key to generate initial state s-boxes
"""

# XOR in key bits to the P array
key_len = len(self.key)
cur_pos = 0
for i in range(len(self.P)):
      if cur_pos+4 >= key_len:
            next_pos = (cur_pos+4)%key_len
            wrapped = self.key[cur_pos:]
            wrapped.extend(self.key[:next_pos])
            self.P[i] ^= unpack('>I',wrapped)[0]
            cur_pos = next_pos
      else:
```

```
                  self.P[i] ^= unpack('>I',self.key[cur_pos:cur_pos+4])[0]
                  cur_pos += 4

      #Encrypt the all-0 string with the algorithm

      all_zero = bytearray.fromhex('00 00 00 00 00 00 00 00')

      #replace each pair of elements in P with the encryption from the
previous 2

      for i in range(0,len(self.P),2):
            all_zero = self.encrypt_block(all_zero)
            self.P[i] = unpack('>I',all_zero[0:4])[0]
            self.P[i+1] = unpack('>I',all_zero[4:8])[0]

      #replace each pair of elements in S with the encryption from the
previous 2
      for i in range(len(self.S)):
            for j in range(0,len(self.S[i]),2):
                  all_zero = self.encrypt_block(all_zero)
                  self.S[i][j] = unpack('>I',all_zero[0:4])[0]
                  self.S[i][j+1] = unpack('>I',all_zero[4:8])[0]



      def feistel(self, num):
      """
      Passes a number through the feistel function
      """
      # ((S1,a + S2,b mod 2^32) XOR S3,c) + S4,d mod 2^32
      # First, divide num into 4 quarters, a, b, c, and d

      a,b,c,c = num & 0xFF, num & 0xFF00 >> 8, num & 0xFF0000 >> 16, num &
0xFF000000 >> 24
      #parts = pack('>I', num)
      #a,b,c,d = parts[0],parts[1],parts[2],parts[3]
      return (((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
                  + self.S[3][d]) % 4294967296



      def encrypt_block(self, block):
      """
      Applies the algorithm to a block
      """
      #separate out the left and right halves
      left = block[3] | (block[2] << 8) | (block[1] << 16) | (block[0] << 24)
      right = block[7] | (block[6] << 8) | (block[5] << 16) | (block[4] << 24)


      #XOR the first subkey with the left
      xleft = left^self.P[0]

      #Split the left half into 4 parts
      d,c,b,a = xleft & 0xFF, (xleft & 0xFF00) >> 8, (xleft & 0xFF0000) >> 16,
(xleft & 0xFF000000) >> 24

      #Calculate the right half in place
      right = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^right^self.P[1]


      d,c,b,a = right & 0xFF, (right & 0xFF00) >> 8, (right & 0xFF0000) >> 16,
(right & 0xFF000000) >> 24
      #calculate the new lwft half in place
      left = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^xleft

      #Rounds 3 & 4
      xleft = left^self.P[2]
      d,c,b,a = xleft & 0xFF, (xleft & 0xFF00) >> 8, (xleft & 0xFF0000) >> 16,
(xleft & 0xFF000000) >> 24

      right = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^right^self.P[3]
      d,c,b,a = right & 0xFF, (right & 0xFF00) >> 8, (right & 0xFF0000) >> 16,
(right & 0xFF000000) >> 24

      left = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^xleft


      #Rounds 5 & 6
      xleft = left^self.P[4]
      d,c,b,a = xleft & 0xFF, (xleft & 0xFF00) >> 8, (xleft & 0xFF0000) >> 16,
(xleft & 0xFF000000) >> 24
```

```
      right = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^right^self.P[5]
      d,c,b,a = right & 0xFF, (right & 0xFF00) >> 8, (right & 0xFF0000) >> 16,
(right & 0xFF000000) >> 24

      left = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^xleft


      #Rounds 7 & 8
      xleft = left^self.P[6]
      d,c,b,a = xleft & 0xFF, (xleft & 0xFF00) >> 8, (xleft & 0xFF0000) >> 16,
(xleft & 0xFF000000) >> 24

      right = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^right^self.P[7]
      d,c,b,a = right & 0xFF, (right & 0xFF00) >> 8, (right & 0xFF0000) >> 16,
(right & 0xFF000000) >> 24

      left = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^xleft


      #Rounds 9 & 10
      xleft = left^self.P[8]
      d,c,b,a = xleft & 0xFF, (xleft & 0xFF00) >> 8, (xleft & 0xFF0000) >> 16,
(xleft & 0xFF000000) >> 24

      right = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^right^self.P[9]
      d,c,b,a = right & 0xFF, (right & 0xFF00) >> 8, (right & 0xFF0000) >> 16,
(right & 0xFF000000) >> 24

      left = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^xleft

      #Rounds 11 & 12
      xleft = left^self.P[10]
      d,c,b,a = xleft & 0xFF, (xleft & 0xFF00) >> 8, (xleft & 0xFF0000) >> 16,
(xleft & 0xFF000000) >> 24

      right = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^right^self.P[11]
      d,c,b,a = right & 0xFF, (right & 0xFF00) >> 8, (right & 0xFF0000) >> 16,
(right & 0xFF000000) >> 24

      left = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^xleft

      #Rounds 13 & 14
      xleft = left^self.P[12]
      d,c,b,a = xleft & 0xFF, (xleft & 0xFF00) >> 8, (xleft & 0xFF0000) >> 16,
(xleft & 0xFF000000) >> 24

      right = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^right^self.P[13]
      d,c,b,a = right & 0xFF, (right & 0xFF00) >> 8, (right & 0xFF0000) >> 16,
(right & 0xFF000000) >> 24

      left = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^xleft

      #Rounds 15 & 16
      xleft = left^self.P[14]
      d,c,b,a = xleft & 0xFF, (xleft & 0xFF00) >> 8, (xleft & 0xFF0000) >> 16,
(xleft & 0xFF000000) >> 24

      right = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^right^self.P[15]
      d,c,b,a = right & 0xFF, (right & 0xFF00) >> 8, (right & 0xFF0000) >> 16,
(right & 0xFF000000) >> 24

      left = ((((self.S[0][a] + self.S[1][b] % 4294967296) ^ self.S[2][c])
            + self.S[3][d]) % 4294967296)^xleft




      #unswap
      left, right = right, left

      #XOR in the last 2 subkeys
      right ^= self.P[16]
      left ^= self.P[17]
      return pack('>Q',left << 32 | right)
```

```
def decrypt_block(self, block):
"""
Un-Applies the algorithm to a block
"""
left = unpack('>I',block[0:4])[0]
right = unpack('>I',block[4:8])[0]

for i in range(17,1,-1):
      left ^= self.P[i]
      right = self.feistel(left) ^ right
      left, right = right, left

left, right = right, left
right ^= self.P[1]
left ^= self.P[0]

ret = bytearray(pack('>I', left))
ret.extend(pack('>I', right))
return ret
```

7. Revised Running Time Measurements

```
Overall Time: 67.798 seconds


  Ordered by: standard name


  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
2500000       1.496  0.000  1.496  0.000 blowfish.py:285(blockSize)
       1      0.000  0.000  0.000  0.000 blowfish.py:292(keySize)
 2500000      6.561  0.000   54.919       0.000 blowfish.py:307(encrypt)
       1      0.001  0.001  0.012  0.012 blowfish.py:323(generate_s_box)
 2500521   46.119     0.000   48.368       0.000 blowfish.py:372(encrypt_block)
       1      0.000  0.000  0.012  0.012 blowfish.py:5(__init__)
       1      2.474  2.474   67.798   67.798 timetrial.py:13(test_ntimes)
 2500000      8.896  0.000   65.312       0.000 timetrial.py:19(test_once)
       1      0.000  0.000   67.798   67.798 {built-in method exec}
       1      0.000  0.000  0.000  0.000 {built-in method fromhex}
       8      0.000  0.000  0.000  0.000 {built-in method len}
 2500521      2.249  0.000  2.249  0.000 {built-in method pack}
       1060   0.001  0.000  0.001  0.000 {built-in method unpack}
       1      0.000  0.000  0.000  0.000 {method 'disable' of '_lsprof.Profiler' objects}
       9      0.000  0.000  0.000  0.000 {method 'extend' of 'bytearray' objects}
Done.


       12502127 function calls in 66.482 seconds


  Ordered by: standard name


  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       1      0.000  0.000   66.482   66.482 <string>:1(<module>)
 2500000      1.434  0.000  1.434  0.000 blowfish.py:285(blockSize)
       1      0.000  0.000  0.000  0.000 blowfish.py:292(keySize)
 2500000      6.341  0.000   53.959       0.000 blowfish.py:307(encrypt)
       1      0.001  0.001  0.012  0.012 blowfish.py:323(generate_s_box)
 2500521   45.641     0.000   47.629       0.000 blowfish.py:372(encrypt_block)
       1      0.000  0.000  0.012  0.012 blowfish.py:5(__init__)
       1      2.236  2.236   66.482   66.482 timetrial.py:13(test_ntimes)
 2500000      8.841  0.000   64.234       0.000 timetrial.py:19(test_once)
       1      0.000  0.000   66.482   66.482 {built-in method exec}
       1      0.000  0.000  0.000  0.000 {built-in method fromhex}
       8      0.000  0.000  0.000  0.000 {built-in method len}
 2500521      1.988  0.000  1.988  0.000 {built-in method pack}
       1060   0.001  0.000  0.001  0.000 {built-in method unpack}
       1      0.000  0.000  0.000  0.000 {method 'disable' of '_lsprof.Profiler' objects}
       9      0.000  0.000  0.000  0.000 {method 'extend' of 'bytearray' objects}
Done.


       12502127 function calls in 70.094 seconds


  Ordered by: standard name


  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       1      0.000  0.000   70.094   70.094 <string>:1(<module>)
 2500000      1.452  0.000  1.452  0.000 blowfish.py:285(blockSize)
       1      0.000  0.000  0.000  0.000 blowfish.py:292(keySize)
 2500000      6.712  0.000   57.675       0.000 blowfish.py:307(encrypt)
       1      0.002  0.002  0.012  0.012 blowfish.py:323(generate_s_box)
 2500521   48.961     0.000   50.973       0.000 blowfish.py:372(encrypt_block)
       1      0.000  0.000  0.012  0.012 blowfish.py:5(__init__)
       1      2.140  2.140   70.094   70.094 timetrial.py:13(test_ntimes)
 2500000      8.815  0.000   67.942       0.000 timetrial.py:19(test_once)
       1      0.000  0.000   70.094   70.094 {built-in method exec}
       1      0.000  0.000  0.000  0.000 {built-in method fromhex}
       8      0.000  0.000  0.000  0.000 {built-in method len}
 2500521      2.011  0.000  2.011  0.000 {built-in method pack}
       1060   0.001  0.000  0.001  0.000 {built-in method unpack}
       1      0.000  0.000  0.000  0.000 {method 'disable' of '_lsprof.Profiler' objects}
       9      0.000  0.000  0.000  0.000 {method 'extend' of 'bytearray' objects}
Done.
```

8. Analysis of Revised Measurements

For our revised implementation, we first halved the number of loop iterations by calculating two rounds at once, which also removed overhead from swapping the left and right halves in memory. Then we unrolled the loop completely and manually inlined all calls to the feistel function. Finally, we replaced calls to struct.pack with manual bitshifts and OR operations whenever packing an unsigned 32-bit integer. A single call to pack remains for packing an unsigned 64-bit integer into bytes.
These revisions provided a 2.94x performance boost,reducing execution time from 189 seconds to 68 seconds. Unrolling the for-loop was helpful, but more significant improvements were gained from the manual bitshift operations and inlining the feistel function.

9. Developer's Manual

This project was developed with Python 3.3, and can be imported with the statement `import blowfish`. To use the cipher, instantiate a copy of the Blowfish class with the key as a bytearray. Then call the encrypt() or decrypt() methods of that object to encrypt or decrypt a block, respectively. Be aware that the order in which blocks are encrypted or decrypted is important and will affect the output of these functions.

Two tests are included; standard test vectors of the blowfish function itself in battery.py and a profiler/time trial in timetrial.py. These tests may be run by cd'ing into the src directory and running them with python3 ../tests/battery.py or python3 ../tests/timetrial.py 2500000 (Where 2500000 is the number of times you want to run the encryption function in the trial.)

10. User's Manual

Included is fishc.py, which encrypts and decrypts whole files (in Electronic Codebook mode). To use it, cd into the src dir and run python3 fishc.py {-e/-d} [key (hex)] [input file] [output file]. Use -e for encryption and -d for decryption.

11. What We Learned

We learned that Python has a philosophy of flexibility over runtime speed, which was especially apparent in the case of struct.pack. The pack function was an order of magnitude slower than the equivalent manual bitshift operations, but was fairly simple to fix. On the plus side, Python's flexibility made it easier to get a working version of Blowfish up and running. Another drawback of Python is the overhead of function calls, which cannot be inlined by the interpreter. Manually inlining these calls in the source code provided a significant performance boost.

12. Possible Future Work

The remaining calls to struct.pack and struct.unpack could be replaced with manual bitshift operations, further helping to optimize the program. The function to generate S-boxes is also ripe for optimization, but it may not be worth the effort as it is only called once.

13. Team Member Responsibilities

Stephen Yingling: Led the team through the algorithm selection process and debugged the initial implementation
Wesley Wigham: Created the GitHub page, stubbed the project structure, and wrote the first draft of code
Chad Zawistowski: Implemented testing functions and created call stack diagrams for the presentation

All team members helped to create the original blowfish implementation as well as to optimize its performance. Documents and presentations were created together and equally divided into speaking parts.

14. References

Bruce Schneier's Original Paper on Blowfish: https://www.schneier.com/paper-blowfish-fse.html
Blowfish Test Vectors: https://www.schneier.com/code/vectors.txt
Blowfish Constants: https://www.schneier.com/code/constants.txt
Figures 2 and 3: http://en.wikipedia.org/wiki/Blowfish_%28cipher%29