```
/*****************************************************************************
*                                                                            *
*                         Blowfish Encryption Algorithm                      *
*                       Copyright Peter Gutmann 1995-1996                    *
*                                                                            *
*****************************************************************************/
#include "stdafx.h"
#include "structs.h"

//  typedef unsigned long       LONG;
// typedef unsigned char    BYTE;
//  typedef unsigned short  WORD;



/* A structure to hold the Blowfish key */
//  in structs.h


/* Prototypes for functions in BLOWFISH.C */

void FAR PASCAL blowfishEncrypt( BLOWFISH_KEY *key, unsigned char *data );
void FAR PASCAL blowfishDecrypt( BLOWFISH_KEY *key, unsigned char *data );
BOOL FAR PASCAL blowfishKeyInit( BLOWFISH_KEY *key, unsigned char *userKey, short userKeyLe
ngth );
//BOOL FAR PASCAL blowfishKeyInitSK( BLOWFISH_KEY *key, unsigned char *userKey, short userK
eyLength, short keySetupIterations );


static unsigned long initialParray[] = {
     608135816UL, 2242054355UL,  320440878UL,   57701188UL,
    2752067618UL,  698298832UL,  137296536UL, 3964562569UL,
    1160258022UL,  953160567UL, 3193202383UL,  887688300UL,
    3232508343UL, 3380367581UL, 1065670069UL, 3041331479UL,
    2450970073UL, 2306472731UL
    };

static unsigned long initialSbox1[] = {
    3509652390UL, 2564797868UL,  805139163UL, 3491422135UL,
    3101798381UL, 1780907670UL, 3128725573UL, 4046225305UL,
     614570311UL, 3012652279UL,  134345442UL, 2240740374UL,
    1667834072UL, 1901547113UL, 2757295779UL, 4103290238UL,
     227898511UL, 1921955416UL, 1904987480UL, 2182433518UL,
    2069144605UL, 3260701109UL, 2620446009UL,  720527379UL,
    3318853667UL,  677414384UL, 3393288472UL, 3101374703UL,
    2390351024UL, 1614419982UL, 1822297739UL, 2954791486UL,
    3608508353UL, 3174124327UL, 2024746970UL, 1432378464UL,
    3864339955UL, 2857741204UL, 1464375394UL, 1676153920UL,
    1439316330UL,  715854006UL, 3033291828UL,  289532110UL,
    2706671279UL, 2087905683UL, 3018724369UL, 1668267050UL,
     732546397UL, 1947742710UL, 3462151702UL, 2609353502UL,
    2950085171UL, 1814351708UL, 2050118529UL,  680887927UL,
     999245976UL, 1800124847UL, 3300911131UL, 1713906067UL,
    1641548236UL, 4213287313UL, 1216130144UL, 1575780402UL,
    4018429277UL, 3917837745UL, 3693486850UL, 3949271944UL,
     596196993UL, 3549867205UL,  258830323UL, 2213823033UL,
     772490370UL, 2760122372UL, 1774776394UL, 2652871518UL,
     566650946UL, 4142492826UL, 1728879713UL, 2882767088UL,
    1783734482UL, 3629395816UL, 2517608232UL, 2874225571UL,
    1861159788UL,  326777828UL, 3124490320UL, 2130389656UL,
    2716951837UL,  967770486UL, 1724537150UL, 2185432712UL,
    2364442137UL, 1164943284UL, 2105845187UL,  998989502UL,
    3765401048UL, 2244026483UL, 1075463327UL, 1455516326UL,
    1322494562UL,  910128902UL,  469688178UL, 1117454909UL,
     936433444UL, 3490320968UL, 3675253459UL, 1240580251UL,
     122909385UL, 2157517691UL,  634681816UL, 4142456567UL,
    3825094682UL, 3061402683UL, 2540495037UL,   79693498UL,
    3249098678UL, 1084186820UL, 1583128258UL,  426386531UL,
    1761308591UL, 1047286709UL,  322548459UL,  995290223UL,
    1845252383UL, 2603652396UL, 3431023940UL, 2942221577UL,
    3202600964UL, 3727903485UL, 1712269319UL,  422464435UL,
    3234572375UL, 1170764815UL, 3523960633UL, 3117677531UL,
    1434042557UL,  442511882UL, 3600875718UL, 1076654713UL,
```

```
    1738483198UL, 4213154764UL, 2393238008UL, 3677496056UL,
    1014306527UL, 4251020053UL,  793779912UL, 2902807211UL,
     842905082UL, 4246964064UL, 1395751752UL, 1040244610UL,
    2656851899UL, 3396308128UL,  445077038UL, 3742853595UL,
    3577915638UL,  679411651UL, 2892444358UL, 2354009459UL,
    1767581616UL, 3150600392UL, 3791627101UL, 3102740896UL,
     284835224UL, 4246832056UL, 1258075500UL,  768725851UL,
    2589189241UL, 3069724005UL, 3532540348UL, 1274779536UL,
    3789419226UL, 2764799539UL, 1660621633UL, 3471099624UL,
    4011903706UL,  913787905UL, 3497959166UL,  737222580UL,
    2514213453UL, 2928710040UL, 3937242737UL, 1804850592UL,
    3499020752UL, 2949064160UL, 2386320175UL, 2390070455UL,
    2415321851UL, 4061277028UL, 2290661394UL, 2416832540UL,
    1336762016UL, 1754252060UL, 3520065937UL, 3014181293UL,
     791618072UL, 3188594551UL, 3933548030UL, 2332172193UL,
    3852520463UL, 3043980520UL,  413987798UL, 3465142937UL,
    3030929376UL, 4245938359UL, 2093235073UL, 3534596313UL,
     375366246UL, 2157278981UL, 2479649556UL,  555357303UL,
    3870105701UL, 2008414854UL, 3344188149UL, 4221384143UL,
    3956125452UL, 2067696032UL, 3594591187UL, 2921233993UL,
       2428461UL,  544322398UL,  577241275UL, 1471733935UL,
     610547355UL, 4027169054UL, 1432588573UL, 1507829418UL,
    2025931657UL, 3646575487UL,  545086370UL,   48609733UL,
    2200306550UL, 1653985193UL,  298326376UL, 1316178497UL,
    3007786442UL, 2064951626UL,  458293330UL, 2589141269UL,
    3591329599UL, 3164325604UL,  727753846UL, 2179363840UL,
     146436021UL, 1461446943UL, 4069977195UL,  705550613UL,
    3059967265UL, 3887724982UL, 4281599278UL, 3313849956UL,
    1404054877UL, 2845806497UL,  146425753UL, 1854211946UL
    };

static unsigned long initialSbox2[] = {
    1266315497UL, 3048417604UL, 3681880366UL, 3289982499UL,
    2909710000UL, 1235738493UL, 2632868024UL, 2414719590UL,
    3970600049UL, 1771706367UL, 1449415276UL, 3266420449UL,
     422970021UL, 1963543593UL, 2690192192UL, 3826793022UL,
    1062508698UL, 1531092325UL, 1804592342UL, 2583117782UL,
    2714934279UL, 4024971509UL, 1294809318UL, 4028980673UL,
    1289560198UL, 2221992742UL, 1669523910UL,   35572830UL,
     157838143UL, 1052438473UL, 1016535060UL, 1802137761UL,
    1753167236UL, 1386275462UL, 3080475397UL, 2857371447UL,
    1040679964UL, 2145300060UL, 2390574316UL, 1461121720UL,
    2956646967UL, 4031777805UL, 4028374788UL,   33600511UL,
    2920084762UL, 1018524850UL,  629373528UL, 3691585981UL,
    3515945977UL, 2091462646UL, 2486323059UL,  586499841UL,
     988145025UL,  935516892UL, 3367335476UL, 2599673255UL,
    2839830854UL,  265290510UL, 3972581182UL, 2759138881UL,
    3795373465UL, 1005194799UL,  847297441UL,  406762289UL,
    1314163512UL, 1332590856UL, 1866599683UL, 4127851711UL,
     750260880UL,  613907577UL, 1450815602UL, 3165620655UL,
    3734664991UL, 3650291728UL, 3012275730UL, 3704569646UL,
    1427272223UL,  778793252UL, 1343938022UL, 2676280711UL,
    2052605720UL, 1946737175UL, 3164576444UL, 3914038668UL,
    3967478842UL, 3682934266UL, 1661551462UL, 3294938066UL,
    4011595847UL,  840292616UL, 3712170807UL,  616741398UL,
     312560963UL,  711312465UL, 1351876610UL,  322626781UL,
    1910503582UL,  271666773UL, 2175563734UL, 1594956187UL,
      70604529UL, 3617834859UL, 1007753275UL, 1495573769UL,
    4069517037UL, 2549218298UL, 2663038764UL,  504708206UL,
    2263041392UL, 3941167025UL, 2249088522UL, 1514023603UL,
    1998579484UL, 1312622330UL,  694541497UL, 2582060303UL,
    2151582166UL, 1382467621UL,  776784248UL, 2618340202UL,
    3323268794UL, 2497899128UL, 2784771155UL,  503983604UL,
    4076293799UL,  907881277UL,  423175695UL,  432175456UL,
    1378068232UL, 4145222326UL, 3954048622UL, 3938656102UL,
    3820766613UL, 2793130115UL, 2977904593UL,   26017576UL,
    3274890735UL, 3194772133UL, 1700274565UL, 1756076034UL,
    4006520079UL, 3677328699UL,  720338349UL, 1533947780UL,
     354530856UL,  688349552UL, 3973924725UL, 1637815568UL,
     332179504UL, 3949051286UL,   53804574UL, 2852348879UL,
    3044236432UL, 1282449977UL, 3583942155UL, 3416972820UL,
    4006381244UL, 1617046695UL, 2628476075UL, 3002303598UL,
    1686838959UL,  431878346UL, 2686675385UL, 1700445008UL,
```

```
    1080580658UL, 1009431731UL,  832498133UL, 3223435511UL,
    2605976345UL, 2271191193UL, 2516031870UL, 1648197032UL,
    4164389018UL, 2548247927UL,  300782431UL,  375919233UL,
     238389289UL, 3353747414UL, 2531188641UL, 2019080857UL,
    1475708069UL,  455242339UL, 2609103871UL,  448939670UL,
    3451063019UL, 1395535956UL, 2413381860UL, 1841049896UL,
    1491858159UL,  885456874UL, 4264095073UL, 4001119347UL,
    1565136089UL, 3898914787UL, 1108368660UL,  540939232UL,
    1173283510UL, 2745871338UL, 3681308437UL, 4207628240UL,
    3343053890UL, 4016749493UL, 1699691293UL, 1103962373UL,
    3625875870UL, 2256883143UL, 3830138730UL, 1031889488UL,
    3479347698UL, 1535977030UL, 4236805024UL, 3251091107UL,
    2132092099UL, 1774941330UL, 1199868427UL, 1452454533UL,
     157007616UL, 2904115357UL,  342012276UL,  595725824UL,
    1480756522UL,  206960106UL,  497939518UL,  591360097UL,
     863170706UL, 2375253569UL, 3596610801UL, 1814182875UL,
    2094937945UL, 3421402208UL, 1082520231UL, 3463918190UL,
    2785509508UL,  435703966UL, 3908032597UL, 1641649973UL,
    2842273706UL, 3305899714UL, 1510255612UL, 2148256476UL,
    2655287854UL, 3276092548UL, 4258621189UL,  236887753UL,
    3681803219UL,  274041037UL, 1734335097UL, 3815195456UL,
    3317970021UL, 1899903192UL, 1026095262UL, 4050517792UL,
     356393447UL, 2410691914UL, 3873677099UL, 3682840055UL
    };

static unsigned long initialSbox3[] = {
    3913112168UL, 2491498743UL, 4132185628UL, 2489919796UL,
    1091903735UL, 1979897079UL, 3170134830UL, 3567386728UL,
    3557303409UL,  857797738UL, 1136121015UL, 1342202287UL,
     507115054UL, 2535736646UL,  337727348UL, 3213592640UL,
    1301675037UL, 2528481711UL, 1895095763UL, 1721773893UL,
    3216771564UL,   62756741UL, 2142006736UL,  835421444UL,
    2531993523UL, 1442658625UL, 3659876326UL, 2882144922UL,
     676362277UL, 1392781812UL,  170690266UL, 3921047035UL,
    1759253602UL, 3611846912UL, 1745797284UL,  664899054UL,
    1329594018UL, 3901205900UL, 3045908486UL, 2062866102UL,
    2865634940UL, 3543621612UL, 3464012697UL, 1080764994UL,
     553557557UL, 3656615353UL, 3996768171UL,  991055499UL,
     499776247UL, 1265440854UL,  648242737UL, 3940784050UL,
     980351604UL, 3713745714UL, 1749149687UL, 3396870395UL,
    4211799374UL, 3640570775UL, 1161844396UL, 3125318951UL,
    1431517754UL,  545492359UL, 4268468663UL, 3499529547UL,
    1437099964UL, 2702547544UL, 3433638243UL, 2581715763UL,
    2787789398UL, 1060185593UL, 1593081372UL, 2418618748UL,
    4260947970UL,   69676912UL, 2159744348UL,   86519011UL,
    2512459080UL, 3838209314UL, 1220612927UL, 3339683548UL,
     133810670UL, 1090789135UL, 1078426020UL, 1569222167UL,
     845107691UL, 3583754449UL, 4072456591UL, 1091646820UL,
     628848692UL, 1613405280UL, 3757631651UL,  526609435UL,
     236106946UL,   48312990UL, 2942717905UL, 3402727701UL,
    1797494240UL,  859738849UL,  992217954UL, 4005476642UL,
    2243076622UL, 3870952857UL, 3732016268UL,  765654824UL,
    3490871365UL, 2511836413UL, 1685915746UL, 3888969200UL,
    1414112111UL, 2273134842UL, 3281911079UL, 4080962846UL,
     172450625UL, 2569994100UL,  980381355UL, 4109958455UL,
    2819808352UL, 2716589560UL, 2568741196UL, 3681446669UL,
    3329971472UL, 1835478071UL,  660984891UL, 3704678404UL,
    4045999559UL, 3422617507UL, 3040415634UL, 1762651403UL,
    1719377915UL, 3470491036UL, 2693910283UL, 3642056355UL,
    3138596744UL, 1364962596UL, 2073328063UL, 1983633131UL,
     926494387UL, 3423689081UL, 2150032023UL, 4096667949UL,
    1749200295UL, 3328846651UL,  309677260UL, 2016342300UL,
    1779581495UL, 3079819751UL,  111262694UL, 1274766160UL,
     443224088UL,  298511866UL, 1025883608UL, 3806446537UL,
    1145181785UL,  168956806UL, 3641502830UL, 3584813610UL,
    1689216846UL, 3666258015UL, 3200248200UL, 1692713982UL,
    2646376535UL, 4042768518UL, 1618508792UL, 1610833997UL,
    3523052358UL, 4130873264UL, 2001055236UL, 3610705100UL,
    2202168115UL, 4028541809UL, 2961195399UL, 1006657119UL,
    2006996926UL, 3186142756UL, 1430667929UL, 3210227297UL,
    1314452623UL, 4074634658UL, 4101304120UL, 2273951170UL,
    1399257539UL, 3367210612UL, 3027628629UL, 1190975929UL,
    2062231137UL, 2333990788UL, 2221543033UL, 2438960610UL,
```

```
    1181637006UL,  548689776UL, 2362791313UL, 3372408396UL,
    3104550113UL, 3145860560UL,  296247880UL, 1970579870UL,
    3078560182UL, 3769228297UL, 1714227617UL, 3291629107UL,
    3898220290UL,  166772364UL, 1251581989UL,  493813264UL,
     448347421UL,  195405023UL, 2709975567UL,  677966185UL,
    3703036547UL, 1463355134UL, 2715995803UL, 1338867538UL,
    1343315457UL, 2802222074UL, 2684532164UL,  233230375UL,
    2599980071UL, 2000651841UL, 3277868038UL, 1638401717UL,
    4028070440UL, 3237316320UL,    6314154UL,  819756386UL,
     300326615UL,  590932579UL, 1405279636UL, 3267499572UL,
    3150704214UL, 2428286686UL, 3959192993UL, 3461946742UL,
    1862657033UL, 1266418056UL,  963775037UL, 2089974820UL,
    2263052895UL, 1917689273UL,  448879540UL, 3550394620UL,
    3981727096UL,  150775221UL, 3627908307UL, 1303187396UL,
     508620638UL, 2975983352UL, 2726630617UL, 1817252668UL,
    1876281319UL, 1457606340UL,  908771278UL, 3720792119UL,
    3617206836UL, 2455994898UL, 1729034894UL, 1080033504UL
    };

static unsigned long initialSbox4[] = {
     976866871UL, 3556439503UL, 2881648439UL, 1522871579UL,
    1555064734UL, 1336096578UL, 3548522304UL, 2579274686UL,
    3574697629UL, 3205460757UL, 3593280638UL, 3338716283UL,
    3079412587UL,  564236357UL, 2993598910UL, 1781952180UL,
    1464380207UL, 3163844217UL, 3332601554UL, 1699332808UL,
    1393555694UL, 1183702653UL, 3581086237UL, 1288719814UL,
     691649499UL, 2847557200UL, 2895455976UL, 3193889540UL,
    2717570544UL, 1781354906UL, 1676643554UL, 2592534050UL,
    3230253752UL, 1126444790UL, 2770207658UL, 2633158820UL,
    2210423226UL, 2615765581UL, 2414155088UL, 3127139286UL,
     673620729UL, 2805611233UL, 1269405062UL, 4015350505UL,
    3341807571UL, 4149409754UL, 1057255273UL, 2012875353UL,
    2162469141UL, 2276492801UL, 2601117357UL,  993977747UL,
    3918593370UL, 2654263191UL,  753973209UL,   36408145UL,
    2530585658UL,   25011837UL, 3520020182UL, 2088578344UL,
     530523599UL, 2918365339UL, 1524020338UL, 1518925132UL,
    3760827505UL, 3759777254UL, 1202760957UL, 3985898139UL,
    3906192525UL,  674977740UL, 4174734889UL, 2031300136UL,
    2019492241UL, 3983892565UL, 4153806404UL, 3822280332UL,
     352677332UL, 2297720250UL,   60907813UL,   90501309UL,
    3286998549UL, 1016092578UL, 2535922412UL, 2839152426UL,
     457141659UL,  509813237UL, 4120667899UL,  652014361UL,
    1966332200UL, 2975202805UL,   55981186UL, 2327461051UL,
     676427537UL, 3255491064UL, 2882294119UL, 3433927263UL,
    1307055953UL,  942726286UL,  933058658UL, 2468411793UL,
    3933900994UL, 4215176142UL, 1361170020UL, 2001714738UL,
    2830558078UL, 3274259782UL, 1222529897UL, 1679025792UL,
    2729314320UL, 3714953764UL, 1770335741UL,  151462246UL,
    3013232138UL, 1682292957UL, 1483529935UL,  471910574UL,
    1539241949UL,  458788160UL, 3436315007UL, 1807016891UL,
    3718408830UL,  978976581UL, 1043663428UL, 3165965781UL,
    1927990952UL, 4200891579UL, 2372276910UL, 3208408903UL,
    3533431907UL, 1412390302UL, 2931980059UL, 4132332400UL,
    1947078029UL, 3881505623UL, 4168226417UL, 2941484381UL,
    1077988104UL, 1320477388UL,  886195818UL,   18198404UL,
    3786409000UL, 2509781533UL,  112762804UL, 3463356488UL,
    1866414978UL,  891333506UL,   18488651UL,  661792760UL,
    1628790961UL, 3885187036UL, 3141171499UL,  876946877UL,
    2693282273UL, 1372485963UL,  791857591UL, 2686433993UL,
    3759982718UL, 3167212022UL, 3472953795UL, 2716379847UL,
     445679433UL, 3561995674UL, 3504004811UL, 3574258232UL,
      54117162UL, 3331405415UL, 2381918588UL, 3769707343UL,
    4154350007UL, 1140177722UL, 4074052095UL,  668550556UL,
    3214352940UL,  367459370UL,  261225585UL, 2610173221UL,
    4209349473UL, 3468074219UL, 3265815641UL,  314222801UL,
    3066103646UL, 3808782860UL,  282218597UL, 3406013506UL,
    3773591054UL,  379116347UL, 1285071038UL,  846784868UL,
    2669647154UL, 3771962079UL, 3550491691UL, 2305946142UL,
     453669953UL, 1268987020UL, 3317592352UL, 3279303384UL,
    3744833421UL, 2610507566UL, 3859509063UL,  266596637UL,
    3847019092UL,  517658769UL, 3462560207UL, 3443424879UL,
     370717030UL, 4247526661UL, 2224018117UL, 4143653529UL,
    4112773975UL, 2788324899UL, 2477274417UL, 1456262402UL,
```

```
    2901442914UL, 1517677493UL, 1846949527UL, 2295493580UL,
    3734397586UL, 2176403920UL, 1280348187UL, 1908823572UL,
    3871786941UL,  846861322UL, 1172426758UL, 3287448474UL,
    3383383037UL, 1655181056UL, 3139813346UL,  901632758UL,
    1897031941UL, 2986607138UL, 3066810236UL, 3447102507UL,
    1393639104UL,  373351379UL,  950779232UL,  625454576UL,
    3124240540UL, 4148612726UL, 2007998917UL,  544563296UL,
    2244738638UL, 2330496472UL, 2058025392UL, 1291430526UL,
     424198748UL,   50039436UL,   29584100UL, 3605783033UL,
    2429876329UL, 2791104160UL, 1057563949UL, 3255363231UL,
    3075367218UL, 3463963227UL, 1469046755UL,  985887462UL
    };



#define zeroise( memory, size )     memset( memory, 0, size )



#define mgetBLong(memPtr)       \
        ( ( ( unsigned long ) memPtr[ 0 ] << 24 ) | ( ( unsigned long ) memPtr[ 1 ] << 16 )
 | \
          ( ( unsigned long ) memPtr[ 2 ] << 8 ) | ( unsigned long ) memPtr[ 3 ] ); \
        memPtr += 4

#define mputBLong(memPtr,data)  \
        memPtr[ 0 ] = ( unsigned char ) ( ( ( data ) >> 24 ) & 0xFF ); \
        memPtr[ 1 ] = ( unsigned char ) ( ( ( data ) >> 16 ) & 0xFF ); \
        memPtr[ 2 ] = ( unsigned char ) ( ( ( data ) >> 8 ) & 0xFF ); \
        memPtr[ 3 ] = ( unsigned char ) ( ( data ) & 0xFF ); \
        memPtr += 4

#define LITTLE_ENDIAN 1

/* LCRNG start value */

#define LCRNG_START             1

/* Test vectors for the LCRNG and Blowfish-SK itself */

#define LCRNG_INITIAL           23312U
#define LCRNG_FINAL             23021U

#define BLOWFISH_PLAINTEXT      "\x00\x00\x00\x00\x00\x00\x00\x00"
#define BLOWFISH_CIPHERTEXT     "\x4D\x38\xE6\x00\x47\x35\x24\xCF"

/* The LCRNG used for the key setup */

#define lcrng(number)   ( unsigned short ) ( ( ( number * 23311L ) + 1 ) % 65533U )

/* Macros to extract 8-bit values a, b, c, d from a 32-bit value.  The cast
   is necessary because some compilers prefer ints as array indices */

#define exta(x)     ( ( short ) ( ( x >> 24 ) & 0xFF ) )
#define extb(x)     ( ( short ) ( ( x >> 16 ) & 0xFF ) )
#define extc(x)     ( ( short ) ( ( x >> 8 ) & 0xFF ) )
#define extd(x)     ( ( short ) ( ( x ) & 0xFF ) )

#define CRYPT_OK            0       /* No error */

/* Internal errors */

#define CRYPT_ERROR         -1      /* Nonspecific error */
#define CRYPT_SELFTEST      -2      /* Failed self-test */
#define CRYPT_NOMEM         -50     /* Out of memory */

/* The f-function */

#define f(data,S1,S2,S3,S4)     \
    ( ( ( S1[ exta( data ) ] + S2[ extb( data ) ] ) ^ S3[ extc( data ) ] ) + \
                                                    S4[ extd( data ) ] )
/* The individual encrypt/decrypt rounds */
```

```
#define oddRoundE(count,P,S1,S2,S3,S4)  L ^= P[ count - 1 ]; \
                                        R ^= f( L, S1, S2, S3, S4 )
#define evenRoundE(count,P,S1,S2,S3,S4) R ^= P[ count - 1 ]; \
                                        L ^= f( R, S1, S2, S3, S4 )
#define oddRoundD(count,P,S1,S2,S3,S4)  L ^= P[ count + 1 ]; \
                                        R ^= f( L, S1, S2, S3, S4 )
#define evenRoundD(count,P,S1,S2,S3,S4) R ^= P[ count + 1 ]; \
                                        L ^= f( R, S1, S2, S3, S4 )

void FAR PASCAL blowfishEncrypt( BLOWFISH_KEY *key, unsigned char *data )
{
    unsigned char *dataPtr = data;
    unsigned long *P = key->P, *S1 = key->S1, *S2 = key->S2, *S3 = key->S3, *S4 = key->S4;
    unsigned long L, R;

    L = mgetBLong( dataPtr );
    R = mgetBLong( dataPtr );

    /* Perform 16 rounds of encryption */
    oddRoundE(   1, P, S1, S2, S3, S4 );
    evenRoundE(  2, P, S1, S2, S3, S4 );
    oddRoundE(   3, P, S1, S2, S3, S4 );
    evenRoundE(  4, P, S1, S2, S3, S4 );
    oddRoundE(   5, P, S1, S2, S3, S4 );
    evenRoundE(  6, P, S1, S2, S3, S4 );
    oddRoundE(   7, P, S1, S2, S3, S4 );
    evenRoundE(  8, P, S1, S2, S3, S4 );
    oddRoundE(   9, P, S1, S2, S3, S4 );
    evenRoundE( 10, P, S1, S2, S3, S4 );
    oddRoundE(  11, P, S1, S2, S3, S4 );
    evenRoundE( 12, P, S1, S2, S3, S4 );
    oddRoundE(  13, P, S1, S2, S3, S4 );
    evenRoundE( 14, P, S1, S2, S3, S4 );
    oddRoundE(  15, P, S1, S2, S3, S4 );
    evenRoundE( 16, P, S1, S2, S3, S4 );

    /* Perform the final XOR's */
    L ^= P[ 16 ];
    R ^= P[ 17 ];

    dataPtr = data;
    mputBLong( dataPtr, R );
    mputBLong( dataPtr, L );
}

void FAR PASCAL blowfishDecrypt( BLOWFISH_KEY *key, unsigned char *data )
{
    unsigned char *dataPtr = data;
    unsigned long *P = key->P, *S1 = key->S1, *S2 = key->S2, *S3 = key->S3, *S4 = key->S4;
    unsigned long L, R;

    R = mgetBLong( dataPtr );
    L = mgetBLong( dataPtr );

    /* Perform 16 rounds of encryption */
    evenRoundD( 16, P, S1, S2, S3, S4 );
    oddRoundD(  15, P, S1, S2, S3, S4 );
    evenRoundD( 14, P, S1, S2, S3, S4 );
    oddRoundD(  13, P, S1, S2, S3, S4 );
    evenRoundD( 12, P, S1, S2, S3, S4 );
    oddRoundD(  11, P, S1, S2, S3, S4 );
    evenRoundD( 10, P, S1, S2, S3, S4 );
    oddRoundD(   9, P, S1, S2, S3, S4 );
    evenRoundD(  8, P, S1, S2, S3, S4 );
    oddRoundD(   7, P, S1, S2, S3, S4 );
    evenRoundD(  6, P, S1, S2, S3, S4 );
    oddRoundD(   5, P, S1, S2, S3, S4 );
    evenRoundD(  4, P, S1, S2, S3, S4 );
    oddRoundD(   3, P, S1, S2, S3, S4 );
    evenRoundD(  2, P, S1, S2, S3, S4 );
    oddRoundD(   1, P, S1, S2, S3, S4 );
```

```
    /* Perform the final XOR's */
    R ^= P[ 1 ];
    L ^= P[ 0 ];

    dataPtr = data;
    mputBLong( dataPtr, L );
    mputBLong( dataPtr, R );
}

/****************************************************************************
*                                                                           *
*                        Blowfish Key Management Routines                   *
*                                                                           *
****************************************************************************/

/* Get the initial values of the P-array and S-boxes from an external file */

/* Various defines needed for the key setup */

#define BLOWFISH_NO_ROUNDS  16

/* Set up a Blowfish S-box */

void FAR PASCAL initSBox( BLOWFISH_KEY *key, unsigned long *Sbox, unsigned char *buffer )
{
    short sBoxIndex;

    for( sBoxIndex = 0; sBoxIndex < 256; sBoxIndex += 2 )
    {
        unsigned char *bufferPtr = buffer;

        blowfishEncrypt( key, buffer );
        Sbox[ sBoxIndex ] = mgetBLong( bufferPtr );
        Sbox[ sBoxIndex + 1 ] = mgetBLong( bufferPtr );
    }
}

/* Set up a Blowfish key */

BOOL FAR PASCAL blowfishKeyInit( BLOWFISH_KEY *key, unsigned char *userKey, short userKeyLe
ngth )
{
    unsigned char buffer[ BLOWFISH_BLOCKSIZE ];
    short keyIndex = 0, i;

    // Set up the initial P-array and S-boxes based on the digits of pi
    memcpy( key->P, initialParray, sizeof( initialParray ) );
    memcpy( key->S1, initialSbox1, sizeof( initialSbox1 ) );
    memcpy( key->S2, initialSbox2, sizeof( initialSbox2 ) );
    memcpy( key->S3, initialSbox3, sizeof( initialSbox3 ) );
    memcpy( key->S4, initialSbox4, sizeof( initialSbox4 ) );

    // XOR the user key bits into the P-array
    for( i = 0; i < BLOWFISH_NO_ROUNDS + 2; i++ )
    {
        unsigned long value = 0L;   // Needed for > 32-bit processors
        short byteIndex;

        // Get 32 bits of user key and XOR them into the P-array
        for( byteIndex = 0; byteIndex < 4; byteIndex++ )
        {
            value = ( value << 8 ) | userKey[ keyIndex++ ];
            keyIndex %= userKeyLength;
        }
        key->P[ i ] = key->P[ i ] ^ value;
    }

    // Encrypt the all-zero string with the initial P-array to get the final
    //   P-array
    memset( buffer, 0, BLOWFISH_BLOCKSIZE );
    for( i = 0; i < BLOWFISH_NO_ROUNDS + 2; i += 2 )
    {
        unsigned char *bufferPtr = buffer;
```

```
        blowfishEncrypt( key, buffer );
        key->P[ i ] = mgetBLong( bufferPtr );
        key->P[ i + 1 ] = mgetBLong( bufferPtr );
    }

    // Continue the process to fill the S-boxes
    initSBox( key, key->S1, buffer );
    initSBox( key, key->S2, buffer );
    initSBox( key, key->S3, buffer );
    initSBox( key, key->S4, buffer );

    return( TRUE );
}

/**************************************************************************
*                                                                         *
*                       Blowfish-SK Key Management Routines               *
*                                                                         *
**************************************************************************/

// Set the P-array and S-boxes from a buffer full of keying material

void FAR PASCAL setKeyData( BLOWFISH_KEY *key, unsigned char *keyBufPtr )
{
    unsigned long *P = key->P, *S1 = key->S1, *S2 = key->S2, *S3 = key->S3, *S4 = key->S4;
    short i;

    /* Move the data from the keybuffer into the P-array and S-boxes,
       converting it to the local endianness in the process.  We set the
       four S-boxes one after the other rather than in parallel for
       compatibility with code which treats them as a single large set of
       S-boxes */
    for( i = 0; i < BLOWFISH_PARRAY_SIZE; i++ )
    {
        P[ i ] = mgetBLong( keyBufPtr );
    }

    for( i = 0; i < BLOWFISH_SBOX_SIZE; i++ )
    {
        S1[ i ] = mgetBLong( keyBufPtr );
    }

    for( i = 0; i < BLOWFISH_SBOX_SIZE; i++ )
    {
        S2[ i ] = mgetBLong( keyBufPtr );
    }

    for( i = 0; i < BLOWFISH_SBOX_SIZE; i++ )
    {
        S3[ i ] = mgetBLong( keyBufPtr );
    }

    for( i = 0; i < BLOWFISH_SBOX_SIZE; i++ )
    {
        S4[ i ] = mgetBLong( keyBufPtr );
    }

}

// Encrypt data in CFB mode

void FAR PASCAL encryptCFB( BLOWFISH_KEY *key, unsigned char *iv, unsigned char *buffer, sh
ort noBytes )
{
    short i, ivCount;

    while( noBytes )
    {
        ivCount = ( noBytes > BLOWFISH_BLOCKSIZE ) ? BLOWFISH_BLOCKSIZE : noBytes;

        // Encrypt the IV
        blowfishEncrypt( key, iv );
```

```
        // XOR the buffer contents with the encrypted IV
        for( i = 0; i < ivCount; i++ )
            buffer[ i ] ^= iv[ i ];

        // Shift the ciphertext into the IV
        memcpy( iv, buffer, ivCount );

        // Move on to next block of data
        noBytes -= ivCount;
        buffer += ivCount;
    }
}

/* Set up a Blowfish-SK key */



/*
short main( void )
{
    unsigned char *plain1 = ( unsigned char * ) "BLOWFISH";
    unsigned char *key1 = ( unsigned char * ) "abcdefghijklmnopqrstuvwxyz";
    unsigned char *cipher1 = ( unsigned char * ) "\x32\x4E\xD0\xFE\xF4\x13\xA2\x03";
    unsigned char *plain2 = ( unsigned char * ) "\xFE\xDC\xBA\x98\x76\x54\x32\x10";
    unsigned char *key2 = ( unsigned char * ) "Who is John Galt?";
    unsigned char *cipher2 = ( unsigned char * ) "\xCC\x91\x73\x2B\x80\x22\xF6\x84";
    BLOWFISH_KEY bfKey;
    unsigned char buffer[ 8 ];

    memcpy( buffer, plain1, 8 );
    if( blowfishKeyInit( &bfKey, key1, strlen( ( char * ) key1 ) ) != CRYPT_OK )
        puts( "Init failed" );
    blowfishEncrypt( &bfKey, buffer );
//  if( memcmp( buffer, cipher1, 8 ) )
//      return( CRYPT_ERROR );
    blowfishDecrypt( &bfKey, buffer );
    if( memcmp( buffer, plain1, 8 ) )
        return( CRYPT_ERROR );
    memcpy( buffer, plain2, 8 );
    if( blowfishKeyInit( &bfKey, key2, strlen( ( char * ) key2 ) ) != CRYPT_OK )
        puts( "Init failed" );
    blowfishEncrypt( &bfKey, buffer );
    if( memcmp( buffer, cipher2, 8 ) )
        return( CRYPT_ERROR );
    blowfishDecrypt( &bfKey, buffer );
    if( memcmp( buffer, plain2, 8 ) )
        return( CRYPT_ERROR );

    return( CRYPT_OK );
}
*/
```