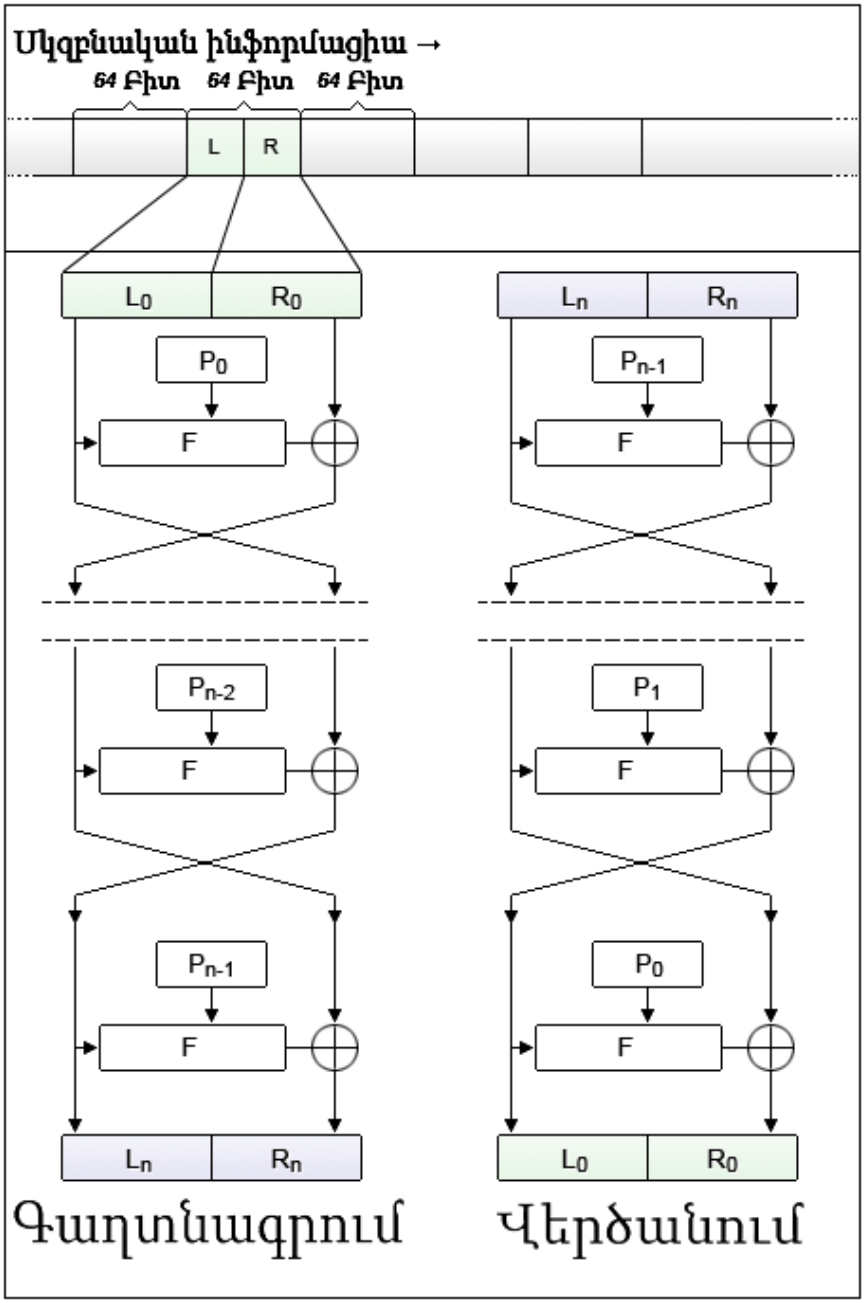
Բովանդակություն

Ներածություն

BlowFish մշակվել է գաղտնագրության և տեղեկատվական անվտանգության ոլորտի անվանի մասնագետ՝ Բրյուս Շնայերի կողմից 1993 թվականին: Ընդհանուր դեպքում, ալգորիթմը բաղկացած է երկու փուլից՝ բանալու ընդլայնումից և տվյալների գաղտնագրումից / վերծանումից: Բանալին պետք է լինի 32-448bits սահմաններում, բլոկների երկարությունը 64bits: Տվյալների գաղտնագրումը տեղի է ունենում Ֆեիստելի ցանցի միջոցով, որն իր հերթին բաղկացած է 16 փուլից: Սկզբում ծանոթանանք թէ ինչ է իրենից ներկայացնում Ֆեիստելի ցանցը:

Ֆեյստելի ցանցը

1971թ. -ին Horst Feistel -ը, IBM Corporation -ում, մշակեց երկու սարք, որոնք իրականացնում էին տարբեր գաղտնագրման ալգորիթմներ, որոնք հետագայում կոչվեցին «Lucifer»: Այս սարքերից մեկում նա օգտագործեց մի սխեմա, որը հետագայում կոչվեց Ֆեյստելի ցանց:



Ֆեյստելի ցանցի սխեմա (Նկ.1)

Ցանցի աշխատելու սկզբունքը բավականին պարզ է.

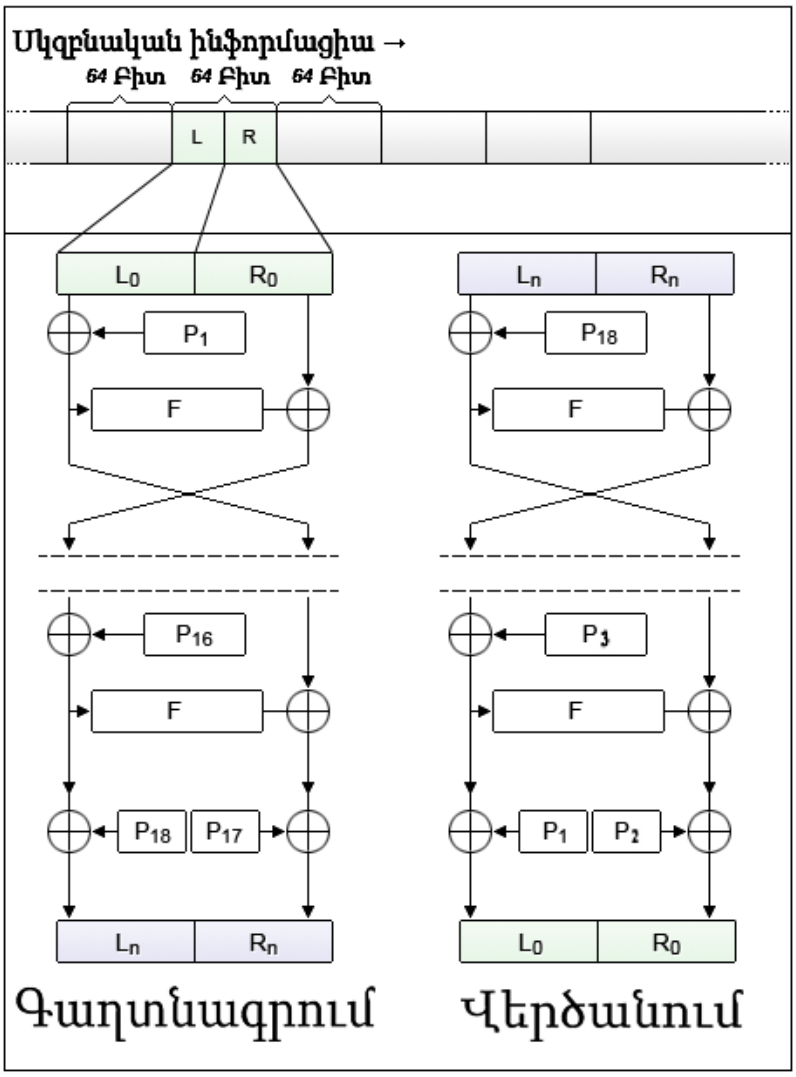
1. Աղբյուրի տվյալները բաժանվում են ֆիքսված երկարության բլոկների (սովորաբար բազմապատիկ երկուսի `64 բիթ, 128 բիթ և այլն): Եթե գաղտնագրվող տվյալների բլոկի երկարությունը բազմապատիկ չէ երկուսի, ապա բլոկը լրացվում է որոշ մեզ հայտնի նշաններով:
2. Մեր ընտրված բլոկները բաժանվում են երկու հավասար ենթաբլոկների `«ձախ» L_0 և «աջ» R_0: 64 -բիթանոց բլոկների դեպքում ձախ և աջ ենթաբլոկները ունենում են 32 բիթ երկարություն:
3. «Ձախ» ենթաբլոկը L_0-ն փոփոխվում է $F(L_0, P_0)$ ֆունկցիայի միջոցով, կախված P_0 բանալուց, որից հետո տրամաբանական բաժանվում է(XOR) «աջ» ենթաբլոկով R_0- ով:
4. Ստացված արդյունքը գրանցվում է նոր ձախ ենթաբլոկում L_1, որը դառնում է հաջորդ փուլի մուտքային տվյալների ձախ կեսը, իսկ «ձախ» ենթաբլոկը L_0 -ն նշանակվում է առանց փոփոխությունների նոր աջ ենթաբլոկ R1 - ում, որը դառնում է աջ կեսը:
5. Այս գործողությունը կրկնվում է n-1 անգամ, մի փուլից մյուսը տեղափոխվելիս, որտեղ փուլային բանալիները փոխվում են (P0, P1, P2 և այլն), որտեղ n -ը փուլերի քանակն է օգտագործված ալգորիթմում:

Գաղտնագրման գործընթացը նման է վերծանման գործընթացին, բացառությամբ այն, որ փուլերի բանալիները օգտագործվում են հակառակ հերթականությամբ:

BlowFish ալգորիթմ

Ընդհանուր առմամբ, Blowfish գաղտնագրման ալգորիթմը իրենից ներկայացնում է Ֆեիստելի ցանցը, բացառությամբ փուլային բանալիների ստեղծելու և օգտագործելու որոշ առանձնահատկություններով (P0, P1 ...): F – ֆունկցիան վերադարձնում է մեզ 32բիտ երկարությամբ բլոկ: 32 -բիտային փուլային բանալիները P_n

1. Հաշվարկվում են ըստ որոշ կանոնների կախված սկզբնական բանալուց (մինչև 448 բիթ երկարություն):
2. Չի հանդիսանում արգումենտ F ֆունկցիայի համար
3. ՈՒղղակիորեն տրամաբանորեն բաժանվում է(XOR) «ձախ» բլոկով: Այս գործողության արդյունքը մուտքային 32-բիթանոց բլոկ է F ֆունկցիայի համար:



BlowFish ալգորիթմի սխեմա (Նկ. 2)

Blowfish ալգորիթմի մեջ գաղտնագրումը տեղի է ունենում 16 փուլով (Ֆեիստելի ցանցի ներսում), իսկ 17-րդ և 18-րդ փուլերի բանալիները տրամաբանորեն բաժանվում են(XOR) վերջին փուլի ձախ և աջ ելքային բլոկներով: Բայց այստեղ հարց է առաջացնում. Եթե օգտագործվում են 18 փուլային բանալիներ, որոնցից յուրաքանչյուրը ունի 32 բիթ երկարություն, ապա վերջում մենք ստանում ենք բանալին 576 բիթ երկարությամբ (18 փուլային բանալի × 32 բիթ): Ինչու՞ է Blowfish -ում բանալու երկարությունը սահմանափակվում 448 բիթով: Պատասխանը շատ պարզ է, այն սահմանափակ չէ: Կարող ենք օգտագործել բանալիներ մինչև 576 բիթ: Սահմանափակումն արվել է ալգորիթմի անվտանգության և գաղտնագրման կայունության պահպանման պահանջների հիման վրա:

BlowFish բանալու ընդլայնում

1. Ընտրում է «անկեղծ թիվ»: Անկեղծ է կոչվում այն թիվը, որն ի սկզբանե չի պարունակում որևէ կրկնող հաջորդականություն և հայտնի է: Blowfish - ում սովորաբար օգտագործում է PI թիվը՝ որպես այդպիսի անկեղծ թիվ: Մենք կօգտագործենք PI թվի համար պատրաստի 16 - ական ներկայացումը(PI թվի համար 8366 16 – ական մանտիսի թիվ http://www.herongyang.com/Cryptography/Blowfish-First-8366-Hex-Digits-of-PI.html):
2. PI թվի մանտիսի արժեքը օգտագործվում է փուլային բանալիների (FIXED_P) և փոխարինման S բլոկների (FIXED_S) ստեղծման համար.

```
const unsigned int FIXED_S[4][256] = {
    {
        0xD1310BA6, 0x98DFB5AC, 0x2FFD72DB, 0xD01ADFB7,
        0xB8E1AFED, 0x6A267E96, 0xBA7C9045, 0xF12C7F99,
        0x24A19947, 0xB3916CF7, 0x0801F2E2, 0x858EFC16,
        0x636920D8, 0x71574E69, 0xA458FEA3, 0xF4933D7E,
        0x0D95748F, 0x728EB658, 0x718BCD58, 0x82154AEE,
        0x7B54A41D, 0xC25A59B5, 0x9C30D539, 0x2AF26013,
        ..............................................
        0xB74E6132, 0xCE77E25B, 0x578FDFE3, 0x3AC372E6
    }
};

const unsigned long FIXED_P[] = {
    0x243F6A88, 0x85A308D3, 0x13198A2E, 0x03707344,
    0xA4093822, 0x299F31D0, 0x082EFA98, 0xEC4E6C89,
    0x452821E6, 0x38D01377, 0xBE5466CF, 0x34E90C6C,
    0xC0AC29B7, 0xC97C50DD, 0x3F84D5B5, 0xB5470917,
    0x9216D5D9, 0x8979FB1B
};
```

pi

```
243F6A8885A308D313198A2E03707344A4093822299F31D0082EFA98EC4E6C89
452821E638D01377BE5466CF34E90C6CC0AC29B7C97C50DD3F84D5B5B5470917
9216D5D98979FB1BD1310BA698DFB5AC2FFD72DBD01ADFB7B8E1AFED6A267E96
BA7C9045F12C7F9924A19947B3916CF70801F2E2858EFC16636920D871574E69
A458FEA3F4933D7E0D95748F728EB658718BCD5882154AEE7B54A41DC25A59B5
9C30D5392AF26013C5D1B023286085F0CA417918B8DB38EF8E79DCB0603A180E
6C9E0E8BB01E8A3ED71577C1BD314B2778AF2FDA55605C60E65525F3AA55AB94
5748986263E8144055CA396A2AAB10B6B4CC5C341141E8CEA15486AF7C72E993
B3EE1411636FBC2A2BA9C55D741831F6CE5C3E169B87931EAFD6BA336C24CF5C
7A325381289586773B8F48986B4BB9AFC4BFE81B6628219361D809CCFB21A991
487CAC605DEC8032EF845D5DE98575B1DC262302EB651B8823893E81D396ACC5
0F6D6FF383F442392E0B4482A484200469C8F04A9E1F9B5E21C66842F6E96C9A
670C9C61ABD388F06A51A0D2D8542F68960FA728AB5133A36EEF0B6C137A3BE4
BA3BF0507EFB2A98A1F1651D39AF017666CA593E82430E888CEE8619456F9FB4
7D84A5C33B8B5EBEE06F75D885C12073401A449F56C16AA64ED3AA62363F7706
1BFEDF72429B023D37D0D724D00A1248DB0FEAD349F1C09B075372C980991B7B
25D479D8F6E8DEF7E3FE501AB6794C3B976CE0BD04C006BAC1A94FB6409F60C4
5E5C9EC2196A246368FB6FAF3E6C53B51339B2EB3B52EC6F6DFC511F9B30952C
CC814544AF5EBD09BEE3D004DE334AFD660F2807192E4BB3C0CBA85745C8740F
D20B5F39B9D3FBDB5579C0BD1A60320AD6A100C6402C7279679F25FEFB1FA3CC
8EA5E9F8DB3222F83C7516DFFD616B152F501EC8AD0552AB323DB5FAFD238760
53317B483E00DF829E5C57BBCA6F8CA01A87562EDF1769DBD542A8F6287EFFC3
AC6732C68C4F5573695B27B0BBCA58C8E1FFA35DB8F011A010FA3D98FD2183B8
4AFCB56C2DD1D35B9A53E479B6F84565D28E49BC4BFB9790E1DDF2DAA4CB7E33
62FB1341CEE4C6E8EF20CADA36774C01D07E9EFE2BF11FB495DBDA4DAE909198
EAAD8E716B93D5A0D08ED1D0AFC725E08E3C5B2F8E7594B78FF6E2FBF2122B64
8888B812900DF01C4FAD5EA0688FC31CD1CFF191B3A8C1AD2F2F2218BE0E1777
EA752DFE8B021FA1E5A0CC0FB56F74E818ACF3D6CE89E299B4A84FE0FD13E0B7
```

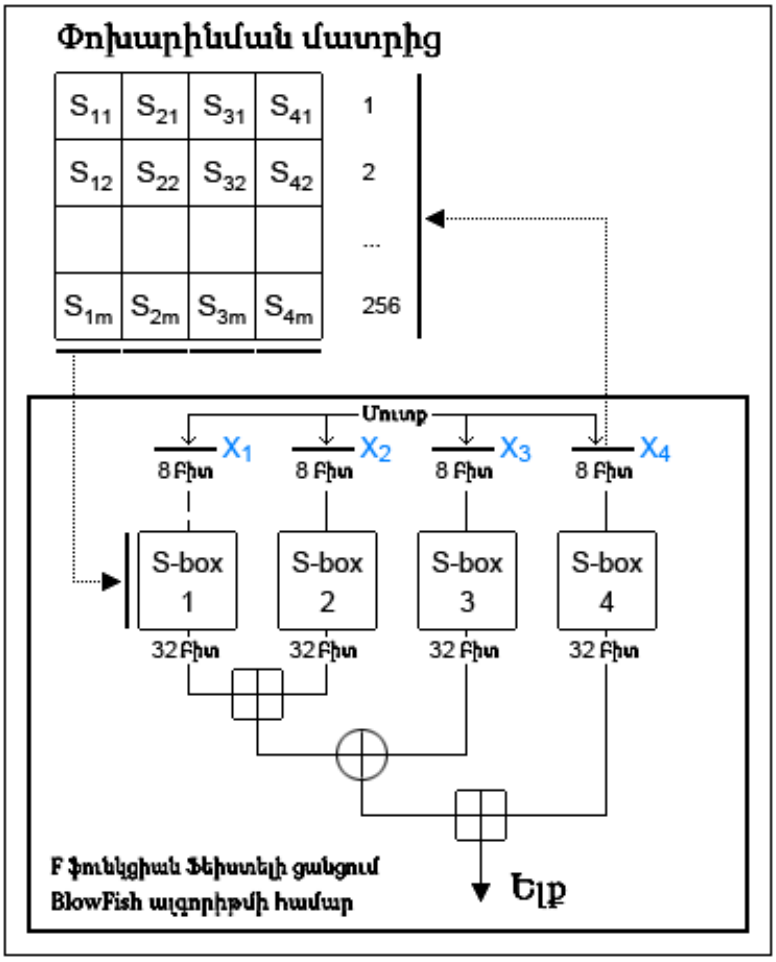
FIXED_P, FIXED_S և PI թվի մանտիսը 16 – ական համակարգում(Նկ. 3)

3. FIXED_P -ի յուրաքանչյուր փուլային բանալու Pn (P_1, P_2 ...) արժեքը տրամաբանորեն բաժանվում է(XOR) ներմուծված բանալու համապատասխան տարրերով K. Օրինակ՝ P_1- ը K առաջին 32 բիտ - ով, P_2- ը ՝ K- ի երկրորդ 32 բիթով և այլն: Եթե ներմուծված բանալին K- ն ավելի կարճ է, քան բոլոր փուլային բանալիների երկարությունը (576 բիթ), ապա այն ավելացվում է իրենով՝ KK, KKK և այլն:

Այսքանով ավարտվում է Blowfish ալգորիթմի բանալու ընդլայնման նախնական փուլը: Բայց նախքան գաղտնագրության / վերծանման անցնելը ցանոթանանք F փուլային ֆունկցիայի հետ:

BlowFish F փուլային ֆունցիա

Փուլային ֆունկցիան շատ պարզ է և օգտագործում է փոխարինման S -բլոկները:



F փուլային ֆունցիա (Նկ. 4)

1. Մուտքային 32 բիթանոց բլոկը բաժանված է չորս 8-բիթանոց բլոկի, եկեք նրանց անվանենք X_1, X_2, X_3, X_4 (Նկ. 4):
2. $S_1[X_1]$ և $S_2[X_2]$ արժեքները ավելացվում են ըստ մոդուլ 2^{32} - ի, ապա արդյունքն տրամաբանական բաժանվում է(XOR) S_3 - ի $[X_3]$ հետ և վերջապես արդյուքն կրկին ավելացվում է ըստ մոդուլ 2^{32} - ի $S_4[X_4]$ - ի հետ:
3. Գործողություններից հետո ստանում ենք F ֆունկցիայի պատասխանը:

F Ֆունկցիայի բանաձևը՝

$$F(X_1, X_2, X_3, X_4) = \Big(\big((S_1[X_1] + S_2[X_2])\ mod\ 2^{32} \oplus S_3[X_3]\big) + S_4[X_4]\Big)\ mod\ 2^{32}$$

BlowFish -ի իրագործումը C# ծրագրավորման լեզվում

```
using System;

namespace BlowFish
{
   class To32Block
   {
      static UInt32[] Sbox0 = { 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 };

static UInt32[] Sbox1 = { 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 };

static UInt32[] Sbox2 = { 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 };

static UInt32[] Sbox3 = { 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 };

static UInt32[] FIXED_P = { 0x243f6a88, 0x85a308d3, 0x13198a2e, 0x03707344, 0xa4093822, 0x299f31d0, 0x082efa98, 0xec4e6c89, 0x452821e6, 0x38d01377, 0xbe5466cf, 0x34e90c6c, 0xc0ac29b7, 0xc97c50dd, 0x3f84d5b5, 0xb5470917, 0x9216d5d9, 0x8979fb1b };

```
static void Main(string[] args)
{
   String key, text;
```

```
    Console.WriteLine("insert key");
    key = Console.ReadLine();
    keyExtension(key);

    Console.WriteLine("insert text");
    text = Console.ReadLine();

    while (text.Length % 8 != 0)
        text += "0";

    byte[] Text8 = new byte[text.Length];
    for (int i = 0; i < text.Length; i++)
        Text8[i] = (byte)text[i];

    PrintArray(Text8);

    Text8 = Encrypt(Text8);
    PrintArray(Text8);

    Text8 = Decrypt(Text8);
    PrintArray(Text8);

    String str = "";
    for (int i = 0; i < Text8.Length; i++)
        Console.Write(Convert.ToChar(Text8[i]));

    Console.ReadKey();
}
```

```
static private byte[] Decrypt(byte[] chiperText)
{
    UInt32[] block32 = F8To32Block(chiperText);

    for (int i = 0; i < block32.Length / 2; i++)
    {
        for (int j = 17; j > 1; j--)
        {
            UInt32[] afterFN = FeistelNetwork(block32[2 * i], block32[2 * i + 1], j);
            block32[2 * i] = afterFN[0];
            block32[2 * i + 1] = afterFN[1];
        }

        UInt32 swap = block32[2 * i];
        block32[2 * i] = block32[2 * i + 1];
        block32[2 * i + 1] = swap;

        block32[2 * i] = block32[2 * i] ^ FIXED_P[0];
        block32[2 * i + 1] = block32[2 * i + 1] ^ FIXED_P[1];
    }
    return F32To8Block(block32);
}

static private byte[] Encrypt(byte[] plainText)
{
    UInt32[] block32 = F8To32Block(plainText);

    for (int i = 0; i < block32.Length / 2; i++)
    {
```

```
        for (int j = 0; j < 16; j++)
        {
            UInt32[] afterFN = FeistelNetwork(block32[2 * i], block32[2 * i + 1], j);
            block32[2 * i] = afterFN[0];
            block32[2 * i + 1] = afterFN[1];
        }

        UInt32 swap = block32[2 * i];
        block32[2 * i] = block32[2 * i + 1];
        block32[2 * i + 1] = swap;

        block32[2 * i] = block32[2 * i] ^ FIXED_P[17];
        block32[2 * i + 1] = block32[2 * i + 1] ^ FIXED_P[16];
    }
    return F32To8Block(block32);
}

static UInt32[] F8To32Block(byte[] block8)
{
    UInt32[] block32 = new UInt32[block8.Length / 4];

    for (int i = 0; i < block32.Length; i++)
    {
        string binary = "";
        for (int j = 0; j < 4; j++)
        {
            String str = "";
            str = str + Convert.ToString(block8[i * 4 + j], 2);
            while (str.Length < 8)
```

```
            str = "0" + str;
            binary = binary + str;
        }

        UInt32 bn = Convert.ToUInt32(binary, 2);
        block32[i] = bn;
    }
    return block32;
}

static byte[] F32To8Block(UInt32[] block32)
{
    byte[] block8 = new byte[block32.Length * 4];

    for (int i = 0; i < block32.Length; i++)
    {
        string binary = Convert.ToString(block32[i], 2);

        while (binary.Length < 32)
            binary = "0" + binary;

        for (int j = 0; j < 4; j++)
            block8[i * 4 + j] = Convert.ToByte(binary.Substring(j * 8, 8), 2);

    }
    return block8;
}
```

```
static private void keyExtension(String bfKey)
{
        byte[] byteKey = new byte[72];
        UInt32[] key32Block;

        //448/8=56
        if (bfKey.Length > 56)
            bfKey = bfKey.Substring(0, 56);

        //576/8 = 72
        while (bfKey.Length < 72)
            bfKey += bfKey;

        bfKey = bfKey.Substring(0, 72);

        for (int i = 0; i < bfKey.Length; i++)
            byteKey[i] = (byte)bfKey[i];

        key32Block = F8To32Block(byteKey);

        for (int i = 0; i < FIXED_P.Length; i++)
            FIXED_P[i] = FIXED_P[i] ^ key32Block[i];
}
```

```
 static private UInt32[] FeistelNetwork(UInt32 leftPiece, UInt32 rightPiece, int index)
{
        UInt32 afLeft;

        leftPiece = leftPiece ^ FIXED_P[index];

        afLeft = FuncF(leftPiece);

        afLeft = afLeft ^ rightPiece;

        rightPiece = leftPiece;
        leftPiece = afLeft;

        return new UInt32[2] { leftPiece, rightPiece };
    }

    static private UInt32 FuncF(UInt32 sboxText)
    {
        byte[] block8;
        UInt32[] block32 = new UInt32[4];
        UInt32 funcEnd;

        block8 = F32To8Block(new UInt32[1] { sboxText });

        block32[0] = Sbox0[block8[0]];
        block32[1] = Sbox1[block8[1]];
        block32[2] = Sbox2[block8[2]];
        block32[3] = Sbox3[block8[3]];
```

```
            funcEnd = (UInt32)(block32[0] + block32[1]);
            funcEnd = funcEnd ^ block32[2];
            funcEnd = (UInt32)(funcEnd + block32[3]);

            return funcEnd;
        }

        static private void PrintArray(byte[] array) {
            for (int i = 0; i < array.Length; i++)
                Console.Write(array[i] + " ");
            Console.WriteLine();
        }
    }
}
```

Եզրակացություն

BlowFish	
Ստեղծման ժամանակը	1993 թ.
Ստեղցել է	Բրյուս Շնայեր (Bruce Schneier)
Ճարտարագիտություն	Ֆեիստելի ցանց
Պարամետր	Բլոկի երկարություն 64 Բանալու երկարություն 32-448 Փուլերի քանակ 16
Առանձնահատկությունները	Անդառնալի փոխարինումների օգտագործումը, ստեղնաշարի փոխարինման հանգույցների կախվածությունը, փոխարինող հանգույցների մեծ չափը (օգտագործվում են փոխարինման 4 հանգույցներ, օգտագործվում են 8-ից 32 բիթ, կախված ստեղնից), ստեղնաշարի փոփոխական չափը 32-ից 448 բիթ է, հիմնական տարրերի առաջացման բարդ սխեմա - հիմնական տարրերի պատրաստում է պահանջում: գաղտնագրման 521 ցիկլերի կատարումը, ինչը էապես բարդացնում է ալգորիթմի վրա դաժան ուժային հարձակումը, բայց այն անհարկի է դարձնում այն համակարգերում օգտագործելու համար, որտեղ բանալին հաճախ փոխվում է, և յուրաքանչյուր ստեղնաշարի վրա կոդավորված են փոքր տվյալներ: Ալգորիթմն ամենալավն է հարմար այն համակարգերի համար, որոնցում տվյալների մեծ հավաքածուն կոդավորված է նույն ստեղնաշարի վրա:

Օգտագործված գրականության ցանկ

1. https://habr.com/ru/post/140394/
2. http://cryptowiki.net/index.php?title=%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_Blowfish
3. https://github.com/Number571/C/blob/master/Cryptography/blowfish.c
4. https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/