ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ
НИЯУ МИФИ

Алгоритмы шифрования семейства blowfish

Иванцов Кирилл Б21-505
Перевощиков Егор Б21-505

Руководитель:
Боронилова Олеся Вячеславовна

АКТУАЛЬНОСТЬ

Безопасность данных в современном мире крайне важна. Для их шифрования было изобретено множество алгоритмов шифрования. Долгое время незаменимым стандартом являлся DES. Но и у него были найдены слабости, из-за чего компании нуждались в новых более надёжных и быстрых алгоритмах шифрования.

ЦЕЛЬ ПРОЕКТА

Спроектировать программы реализующие алгоритмы шифрования семейства blowfish (blowfish, twofish, treefish) и проанализировать их особенности.

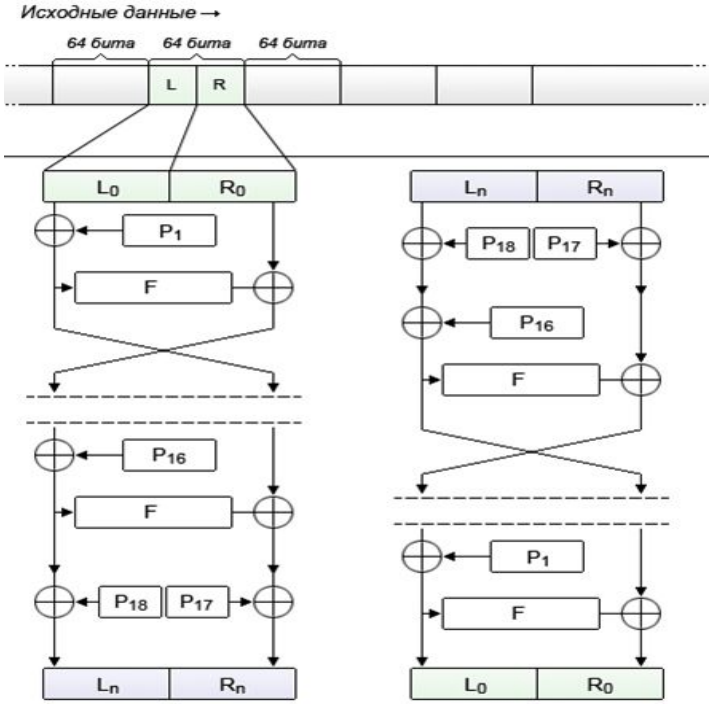
ЗАДАЧИ

1) Изучить принцип работы алгоритмов семейства blowfish.
2) По документации написать программы, реализующую алгоритмы на языке C.
3) Проверить правильность работы написанных алгоритмов, сравнив полученные данные с тестовыми векторами.
4) Проанализировать результаты программ.

Сравнение алгоритмов

алгоритм	Размер блока	Кол-во раундов	Размер ключа	Год создания	процессор	тип
DES	64 бит	16	56 бит	1977	32 бит	Сеть Фейстеля
blowfish	64 бит	16	32-448 бит	1993	32 бит	Сеть Фейстеля
twofish	128 бит	16	128бит 192бит 256бит	1998	32 бит	Сеть Фейстеля
treefish	256 бит 512 бит 1024 бит	72 72 80	256 бит 512 бит 1024 бит	2008	64 бит	Подстановочно-перестановочная сеть

Шифрование и расшифрование blowfish



16 раундов сети Фейстеля.
Для расшифрования подключи берутся от последнего к первому

Сеть Фейстеля blowfish

```
264  uint64_t fastly(uint64_t number)
265  {
266      uint32_t left_number, right_number;
267      right_number = number;
268      left_number = (number >> 32);
269      for (int i = 0; i < 16; i++)
270      {
271          left_number ^= dynamic_keys[i];
272          right_number = foo(left_number) ^ right_number;
273          swap(&left_number, &right_number);
274      }
275      swap(&left_number, &right_number);
276      right_number ^= dynamic_keys[16];
277      left_number ^= dynamic_keys[17];
278      return ((uint64_t)left_number << 32) | right_number;
279  }
```

1. 64-х битовый блок разбивается на два 32-х битовых
2. Левый блок ксорится со значением подключа, проходит через функцию F(), ксорится со значением правого блока
3. Левый и правый блоки меняются

Инициализация подключей blowfish

```
221       for (int kol = 0; kol <= 17; kol++)
222       {
223           xor_key_32 = 0;
224           for (int k = 24; k >= 0; k -= 8)
225           {
226               char a = key8[i];
227               xor_key_32 |= (uint32_t)key8[i] << k;
228               i += 1;
229               if (i == len_key)
230               {
231                   i = 0;
232               }
233           }
```

```
243       } uint64_t plpr = fastly(0), c;
244
245       for (int i = 0; i < 18; i += 2)
246       {
247           dynamic_keys[i] = (plpr >> 32);
248           dynamic_keys[i + 1] = plpr;
249           c = (uint64_t)dynamic_keys[i] << 32 | dynamic_keys[i + 1];
250           plpr = fastly(c);
```

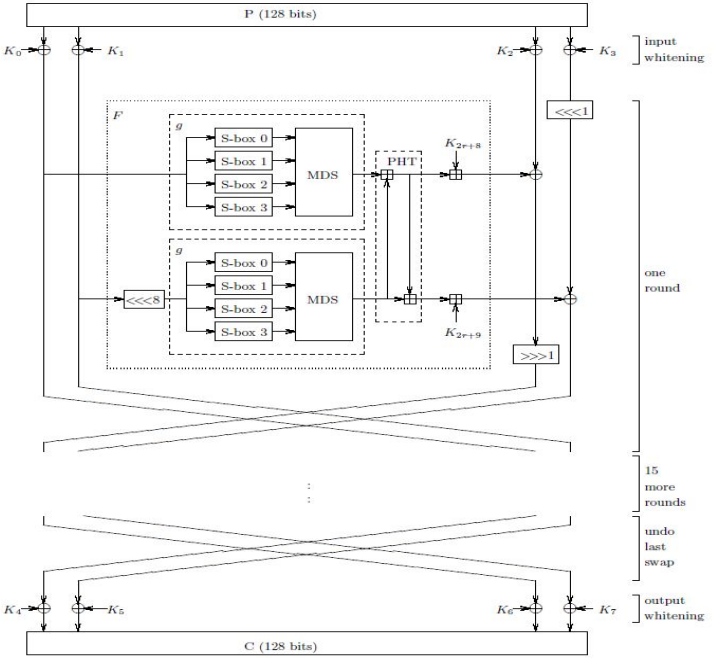
Инициализация ключей проходит в два этапа:

1. Производится операция xor исходных подключей с введенным ключом .

2. Полученные подключи шифруются алгоритмом blowfish.

Функция шифрования twofish

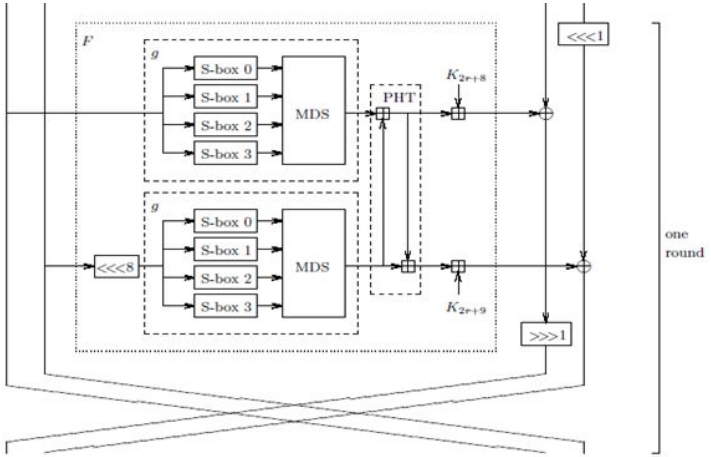
```
void twofish(uint8_t * block128){
        uint32_t block32[4] = {0};
        split128_32(block128, block32);

        whitening_start(block32);

        for(int round = 0; round < 16; round++){
                faistly(block32, round);
        }

        swap(&block32[0], &block32[2]);
        swap(&block32[1], &block32[3]);

        whitening_end(block32);


        return;
}
```



Сеть Фейстеля и Функция F() twofish

```
void faistly(uint32_t *block32, int round){

        block32[3] = ROL(block32[3], 1);
        F(&block32[0], &block32[1], round);
        block32[2] ^= block32[0];
        block32[3] ^= block32[1];
        block32[2] = ROR(block32[2], 1);

        swap(&block32[0], &block32[2]);
        swap(&block32[1], &block32[3]);

        return;
}
```

```
void F(uint32_t *L32, uint32_t *R32, int r){
        g(L32);
        *R32 = ROL(*R32, 8);
        g(R32);
        uint32_t tmp = *L32;
        *L32 = *L32 + *R32 + K[2*r+8];
        *R32 = tmp + *R32 * 2 + K[2*r+9];
        return;
}
```

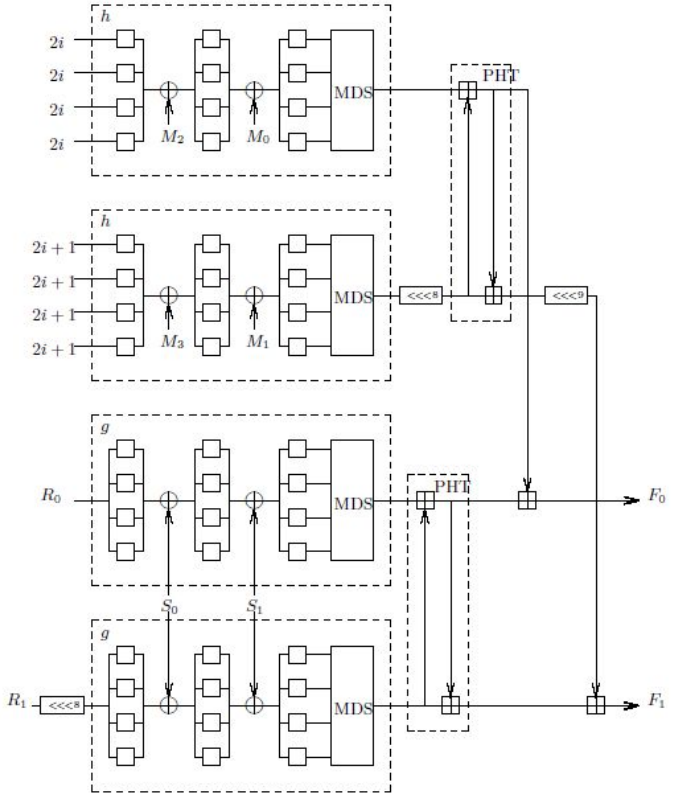


$$
\begin{aligned}
T_0 &= g(R_0) \\
T_1 &= g(\mathrm{ROL}(R_1, 8)) \\
F_0 &= (T_0 + T_1 + K_{2r+8}) \bmod 2^{32} \\
F_1 &= (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}
\end{aligned}
$$

Инициализация подключей twofish

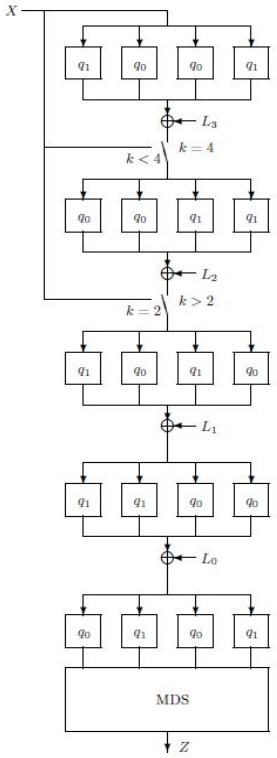
```
uint8_t q(int index, uint8_t x){

        uint8_t a[4];
        uint8_t b[4];
        uint8_t y;

        a[0] = x >> 4;
        b[0] = (x << 4) >> 4;
        a[1] = a[0] ^ b[0] % 16;
        b[1] = ( a[0] ^ ROR4(b[0], 1) ^ (a[0] << 3) ) % 16;
        a[2] = t[index][0][a[1]];
        b[2] = t[index][1][b[1]];
        a[3] = (a[2] ^ b[2]) % 16;
        b[3] = ( a[2] ^ ROR4(b[2], 1) ^ (a[2] << 3) ) % 16;
        a[4] = t[index][2][a[3]];
        b[4] = t[index][3][b[3]];
        y = (b[4] << 4) | a[4];

        return y;
}
```

```
        for(int i = 0; i < 18; i++){
                A = h(2*i*p, Me);
                B = ROL(h((2*i+1)*p, Mo), 8);

                K[2*i] = A + B;
                K[2*i+1] = ROL(A + 2 * B, 9);
        }

        return;
}
```



Функция h() в инициализации ключей twofish

```
uint32_t h(uint32_t X, uint32_t *L){
        uint8_t x[4];
        uint8_t l[4];
        split32_8(X, x);

        split32_8(L[3], l);
        x[0] = q(1, x[0]);
        x[1] = q(0, x[1]);
        x[2] = q(0, x[2]);
        x[3] = q(1, x[3]);
        for(int i = 0; i < 4; i++){
                x[i] ^= l[i];
        }

        split32_8(L[2], l);
        x[0] = q(0, x[0]);
        x[1] = q(0, x[1]);
        x[2] = q(1, x[2]);
        x[3] = q(1, x[3]);
        for(int i = 0; i < 4; i++){
                x[i] ^= l[i];
        }

        split32_8(L[1], l);
        x[0] = q(1, x[0]);
        x[1] = q(0, x[1]);
        x[2] = q(1, x[2]);
        x[3] = q(0, x[3]);
        for(int i = 0; i < 4; i++){
                x[i] ^= l[i];
        }

        split32_8(L[0], l);
        x[0] = q(1, x[0]);
        x[1] = q(1, x[1]);
        x[2] = q(0, x[2]);
        x[3] = q(0, x[3]);
        for(int i = 0; i < 4; i++){
                x[i] ^= l[i];
        }
```

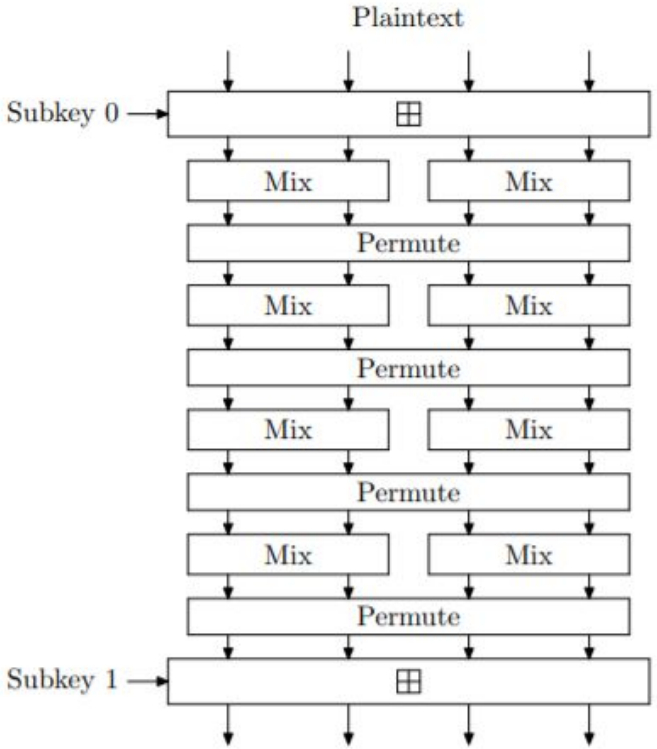


[redacted]

```
uint8_t z[4];
uint32_t Z = 0;
for(int i = 0; i < 4; i++){
        z[i] = MDS[i][0] * x[0] + MDS[i][1] * x[1] + MDS[i][2] * x[2] + MDS[i][3] * x[3];
}

for(int i = 0; i < 4; i++){
        Z |= (uint32_t)z[i] << ((3 - i) * 8);
}

return Z;
```

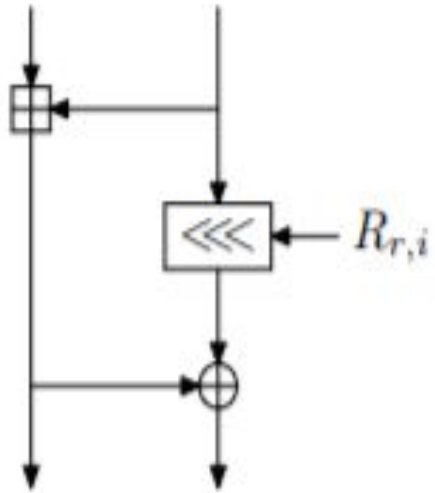
Функция шифрования treefish

```
for(int round = 0; round < Nr; round++){
        if(round % 4 == 0){
                s = round / 4;
                for(int i = 0; i < Nw; i++){
                        e[i] = v[i] + subKey[s][i];
                }
        }
        else{
                for(int i = 0; i < Nw; i++){
                        e[i] = v[i];
                }
        }

        for(int i = 0; i < Nw / 2; i++){
                x0 = e[i*2];
                x1 = e[i*2+1];

                mix(&x0, &x1, set.r[round % 8][i]);

                f[i * 2] = x0;
                f[i * 2 + 1] = x1;
        }


        for(int i = 0; i < Nw; i++){
                v[i] = f[set.pe[i]];
        }
}

for(int i = 0; i < Nw; i++){
        v[i] = v[i] + subKey[Nr/4][i];
}
```

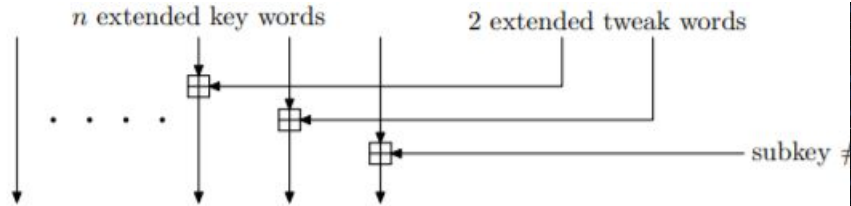


Функции mix() и demix() treefish

```c
void mix(uint64_t *x0, uint64_t *x1, uint64_t R){
        *x0 += *x1;
        *x1 = (*x1 << R) | (*x1 >> (64 - R));
        *x1 ^= *x0;
}

void demix(uint64_t *x0, uint64_t *x1, uint64_t R){
        *x1 ^= *x0;
        *x1 = (*x1 << (64 - R)) | (*x1 >> R);
        *x0 -= *x1;
}
```



Инициализация подключей




```
uint64_t kNw = kNw = 6148914691236517205;
for(int i = 0; i < Nw; i++){
        kNw ^= K[i];
        key[i] = K[i];
}
key[Nw] = kNw;

for(int round = 0; round <= Nr / 4; round++){
        for(int index = 0; index < Nw; index++){
                subKey[round][index] = key[(round + index) % (Nw + 1)];
                if(index == Nw - 3){
                        subKey[round][index] += tweak[round % 3];
                }
                if(index == Nw - 2){
                        subKey[round][index] += tweak[(round + 1) % 3];
                }
                if(index == Nw - 1){
                        subKey[round][index] += round;
                }
        }
        print64("subk ", subKey[round], set.Nw);
}
```

```
выберите длинну блока:
1) 256 бит.
2) 512 бит.
3) 1024 бит.
выбор режима: 2
key:
6b8b4567   327b23c6   643c9869   66334873   74b0dc51   19495cff   2ae8944a   625558ec
tweak:
238e1f29   46e87ccd   3d1b58ba
subk
6b8b4567   327b23c6   643c9869   66334873   74b0dc51   3cd77c28   71d11117   625558ec
subk
327b23c6   643c9869   66334873   74b0dc51   19495cff   71d11117   9f70b1a6   555555552beeafe7
subk
643c9869   66334873   74b0dc51   19495cff   2ae8944a   9f70b1a6   555555554f7ccf0f   6b8b4569
subk
66334873   74b0dc51   19495cff   2ae8944a   625558ec   555555554f7ccf0f   b273c234   327b23c9
subk
74b0dc51   19495cff   2ae8944a   625558ec   555555552beeafe6   b273c234   6f967c80   643c986d
subk
19495cff   2ae8944a   625558ec   555555552beeafe6   6b8b4567   6f967c80   87cab792   66334878
subk
2ae8944a   625558ec   555555552beeafe6   6b8b4567   327b23c6   87cab792   ad1bc540   74b0dc57
subk
625558ec   555555552beeafe6   6b8b4567   327b23c6   643c9869   ad1bc540   b1cc350b   19495d06
subk
555555552beeafe6   6b8b4567   327b23c6   643c9869   66334873   b1cc350b   3cd77c28   2ae89452
subk
6b8b4567   327b23c6   643c9869   66334873   74b0dc51   3cd77c28   71d11117   625558f5
subk
327b23c6   643c9869   66334873   74b0dc51   19495cff   71d11117   9f70b1a6   555555552beeaff0
subk
643c9869   66334873   74b0dc51   19495cff   2ae8944a   9f70b1a6   555555554f7ccf0f   6b8b4572
subk
66334873   74b0dc51   19495cff   2ae8944a   625558ec   555555554f7ccf0f   b273c234   327b23d2
subk
74b0dc51   19495cff   2ae8944a   625558ec   555555552beeafe6   b273c234   6f967c80   643c9876
subk
19495cff   2ae8944a   625558ec   555555552beeafe6   6b8b4567   6f967c80   87cab792   66334881
subk
2ae8944a   625558ec   555555552beeafe6   6b8b4567   327b23c6   87cab792   ad1bc540   74b0dc60
subk
625558ec   555555552beeafe6   6b8b4567   327b23c6   643c9869   ad1bc540   b1cc350b   19495d0f
subk
555555552beeafe6   6b8b4567   327b23c6   643c9869   66334873   b1cc350b   3cd77c28   2ae8945b
subk
6b8b4567   327b23c6   643c9869   66334873   74b0dc51   3cd77c28   71d11117   625558fe
```

Шифрование блоков алгоритмом blowfish.

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	2459468857543 69A
FEDCBA9876543210	FFFFFFFFFFFFFFFF	6B5C5A9C5D9E0A5A

Значения, полученные программой соответствуют значениям тестовых векторов. Алгоритм работает верно.

```
trykirill@LAPTOP-56PTIO9N:~$ ./test
block64                 encription64            decryption64
               0        4ef997456198dd78                       0
ffffffffffffffff        51866fd5b85ecb8a       ffffffffffffffff
1000000000000000        bbfa3644b3ed9850       1000000000000000
1111111111111110        efc4a4fc0a804d97       1111111111111110
1111111111111110        6ad3bc17e8d6f7d4       1111111111111110
 123456789abcdef        dbaa0d7ec492b679        123456789abcdef
               0        4ef997456198dd78                       0
 123456789abcdef         aceab0fc6a0a28d        123456789abcdef
 1a1d6d039776742        59c68245eb05282b        1a1d6d039776742
5cd54ca83def57da        b1b8cc0b250f09a0       5cd54ca83def57da
 248d43806f67172        1730e5778bea1da4        248d43806f67172
51454b582ddf440a        a25e7856cf2651eb       51454b582ddf440a
42fd443059577fa2        353882b109ce8f1a       42fd443059577fa2
 59b5e0851cf143a        48f4d0884c379918        59b5e0851cf143a
 756d8e0774761d2        432193b78951fc98        756d8e0774761d2
762514b829bf486a        13f04154d69d1ae5       762514b829bf486a
3bdd119049372802        2eedda93ffd39c79       3bdd119049372802
26955f6835af609a        d887e0393c2da6e3       26955f6835af609a
164d5e404f275232        5f99d04f5b163969       164d5e404f275232
6b056e18759f5cca        4a057a3b24d3977b       6b056e18759f5cca
  4bd6ef09176062        452031c1e4fada8e         4bd6ef09176062
480d39006ee762f2        7555ae39f59b87bd       480d39006ee762f2
437540c8698f3cfa        53c55f9cb49fc019       437540c8698f3cfa
 72d43a077075292        7a8e7bfa937e89a3        72d43a077075292
 2fe55778117f12a        cf9c5d7a4986adb5        2fe55778117f12a
1d9d5c5018f728c2        d1abb290658bc778       1d9d5c5018f728c2
305532286d6f295a        55cb3774d13ef201       305532286d6f295a
 123456789abcdef        fa34ec4847b268b2        123456789abcdef
 123456789abcdef        a790795108ea3cae        123456789abcdef
 123456789abcdef        c39e072d9fac631d        123456789abcdef
ffffffffffffffff         14933e0cdaff6e4       ffffffffffffffff
               0        f21e9a77b71c49bc                       0
               0        2459468857543 69a                      0
ffffffffffffffff        6b5c5a9c5d9e0a5a       ffffffffffffffff
```

Пример шифрования twofish (128 бит ключ)

```
round -1:   64636261 68676665 6C6B6A69 706F6E6D
round  0:   36A62FBF 79970408 10C7F723 3D7424C7
round  1:   8CEB6EA5 45ADC921 36A62FBF 79970408
round  2:   C7217922 8244482E 8CEB6EA5 45ADC921
round  3:   3CEEBB73 97F9F813 C7217922 8244482E
round  4:   2573A6DB 7B6EA0C 3CEEBB73 97F9F813
round  5:   AECD3634 40EEA3E1 2573A6DB 7B6EA0C
round  6:   822C0EB8 8981861B AECD3634 40EEA3E1
round  7:   86584FF9 D07DA68E 822C0EB8 8981861B
round  8:   6C597BFE D4C9BBE5 86584FF9 D07DA68E
round  9:   F916F82F 6BD83733 6C597BFE D4C9BBE5
round 10:   BB64816A A3477FFD F916F82F 6BD83733
round 11:   F477ADBD 678AB04F BB64816A A3477FFD
round 12:   D684A8A2 8DF12F98 F477ADBD 678AB04F
round 13:   25D3A5FE 8D0A473 D684A8A2 8DF12F98
round 14:   AFC23EF2 68E33B7C 25D3A5FE 8D0A473
round 15:   1BEF70A8 73E8E9E3 AFC23EF2 68E33B7C
round 16:   191BF3F2 8967DD39 1BEF70A8 73E8E9E3
round 17:   AC574AB8 6D95E208 F787C7ED 468696DD
```

```
round 17:   AC574AB8 6D95E208 F787C7ED 468696DD
round 16:   1BEF70A8 73E8E9E3 191BF3F2 8967DD39
round 15:   AFC23EF2 68E33B7C 1BEF70A8 73E8E9E3
round 14:   25D3A5FE 8D0A473 AFC23EF2 68E33B7C
round 13:   D684A8A2 8DF12F98 25D3A5FE 8D0A473
round 12:   F477ADBD 678AB04F D684A8A2 8DF12F98
round 11:   BB64816A A3477FFD F477ADBD 678AB04F
round 10:   F916F82F 6BD83733 BB64816A A3477FFD
round  9:   6C597BFE D4C9BBE5 F916F82F 6BD83733
round  8:   86584FF9 D07DA68E 6C597BFE D4C9BBE5
round  7:   822C0EB8 8981861B 86584FF9 D07DA68E
round  6:   AECD3634 40EEA3E1 822C0EB8 8981861B
round  5:   2573A6DB 7B6EA0C AECD3634 40EEA3E1
round  4:   3CEEBB73 97F9F813 2573A6DB 7B6EA0C
round  3:   C7217922 8244482E 3CEEBB73 97F9F813
round  2:   8CEB6EA5 45ADC921 C7217922 8244482E
round  1:   36A62FBF 79970408 8CEB6EA5 45ADC921
round  0:   10C7F723 3D7424C7 36A62FBF 79970408
round -1:   64636261 68676665 6C6B6A69 706F6E6D
```

Примеры шифрования алгоритмом Treefish (512 бит блок)

```
text:
507ed7ab   2eb141f2   41b71efb   79e2a9e3   7545e146   515f007c   5bd062c2   12200854
encryption text:
22a8d9d77923aab6   a328c3cf21a3e8b7   888cda567e4c10da   b3961df0664b8543   d51e576f3045a193   197068ed9bb85da   eb63813d0da49397   8520bb1eb25a5be2
time : 2367
decryption text:
507ed7ab   2eb141f2   41b71efb   79e2a9e3   7545e146   515f007c   5bd062c2   12200854
```

```
text:
ffffffffd74ddd3b   ffffffff8af97d7c   5940d8ca   7fd272ad   9f5dfea   ffffffffd31127ec   ffffffff9ea0244b   749ea4d8
encryption text:
d66bbf968e842a1b   99679b2eced7144a   ee102d9f1bd4d3ba   cc8097b149aceed   84c35c70a8440dc   6d3dce24b47b1542   e0f9f2f23142a31f   a577ff248dd43f1d
time : 2038
decryption text:
ffffffffd74ddd3b   ffffffff8af97d7c   5940d8ca   7fd272ad   9f5dfea   ffffffffd31127ec   ffffffff9ea0244b   749ea4d8
```

Особенности алгоритмов

Blowfish – быстро работает, но генерация подключей занимает большое количество времени.
Twofish – отличается большей степенью защищённости, но при этом функции шифрования заметно усложняются.
Treefish – множественное повторение простых функций, отсутствие инициализации S блоков,что делает данный алгоритм наиболее быстрым и простым из трёх, но и самым защищённым.

Итог

Проведя данный проект мы познакомились с методами блочного симметричного шифрования, на практике разобрали работу бинарных операций и функций, разобрали сложение и умножение в поле Галуа, которое использовалось в twofish, получили большое кол-во практики написания кода при реализации алгоритмов на языке си. все поставленные задачи были выполнены.

Ссылка на GitHub:

https://github.com/trykirill/blowfish

Ссылки на документацию:

https://www.schneier.com/paper-twofish-paper.html

https://www.schneier.com/wp-content/uploads/2016/02/paper-twofish-paper.pdf

http://www.schneier.com/skein.pdf