Донецький Національний Технічний Університет

Лабораторна робота № 2

«Дослідження сучасних стандартів шифрування»

Виконав:

ст. групи ІПЗІм -17

Лисенко А. С.

Перевірила:

доцент каф. ПМІ

Маслова Н. О.

Покровськ 2017

Вариант 9

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Вар.** | **Довжина ключа шифру «Калина»** | **Розмір блоку** | **Алгоритми для порівняння** | | |
| **1** | **2** | **3** |
| **9** | **128** | **128** | Threefish-256 | IDEA | Гост28147-89 |

Вступ

Стандарти шифрування даних - сукупність правил, використовуваних в потужній алгоритмічної техніці кодуванні інформації

Калина.

При розробці національного стандарту ДСТУ 7624 до: 2014 (блочний шифр «Калина» та режими його роботи) було прийнято рішення забезпечити прозорість проектування і використовувати консервативний підхід із застосуванням добре досліджених конструкцій, що забезпечують запас стійкості для безпечного застосування алгоритму в умовах істотного прогресу крипто-аналітичних технік і засобів обробки даних.  
Новий національний стандарт підтримує розмір блоку і довжину ключа шифрування 128, 256 і 512 біт (довжина ключа дорівнює розміру блоку або в два рази перевищує його), забезпечуючи нормальний, високий і надвисокий рівень стійкості (зараз це єдиний в світі стандарт блочного шифрування, що підтримує 512 - бітові симетричні ключі). Різні варіанти забезпечують гнучкість вибору параметрів для розробників систем криптографічного захисту, що дозволяє отримати як найвищий рівень швидкодії, так і найбільший запас стійкості перетворення.

Стан шифру описується матрицею 8 – х с елементів розширеного двійкового поля *GF(28)*,сформованого незвідним поліномом x8 + x4 + x3 + x2 + 1.

Threefish-256.  
Threefish має дуже просту структуру і може бути використаний для заміни алгоритмів блочного шифрування, будучи швидким і гнучким шифром, що працюють в довільному режимі шифрування. Threefish S-блоки не використовує, заснований на комбінації інструкцій виключає або, складання і циклічного зсуву. Реалізований у вигляді підстановлювальної-перестановної мережі на оборотних операціях, не будучи шифром мережі Фейстеля. Алгоритм передбачає використання tweak-значення, свого роду вектора ініціалізації, дозволяючи змінювати таким чином значення виходу, без зміни ключа, що має позитивний ефект як для реалізації нових режимів шифрування, так і на крипто-стійкості алгоритму.

Як результат думки авторів, що кілька складних раундів часто гірше застосування великого числа простих раундів, алгоритм має нетрадиційно велика кількість раундів - 72 або 80 при ключі 1024 біт

Ядром шифру є проста функція «MIX», що перетворює два 64-бітних без-знакові числа, в процесі якої відбувається складання, циклічний зсув (ROL / ROR), і додавання по модулю 2 (XOR).  
Визначимо функцію шифрування *E (K, T, P),* де:

* *K* - ключ шифрування, рядок довжиною 32, 64, або 128 байт (256, 512, або 1024 біт).
* *T* - tweak-значення, рядок довжиною 16 байт (128 біт).
* *P* - відкритий текст для шифрування, рядок довжиною рівної розміру блоку.

IDEA.

Так як IDEA використовує 128-бітний ключ і 64-бітний розмір блоку, відкритий текст розбивається на блоки по 64 біт. Якщо таке розбиття неможливо, останній блок доповнюється різними способами певною послідовністю біт. Для уникнення витоку інформації про кожну окрему блоці використовуються різні режими шифрування. Кожен вихідний незашифрований 64-бітний блок ділиться на чотири під блоку по 16 біт кожен, так як всі операції алгебри, що використовуються в процесі шифрування, відбуваються над 16-бітними числами. Для шифрування і розшифрування IDEA використовує один і той же алгоритм.

Фундаментальним нововведенням в алгоритмі є використання операцій з різних алгебраїчних груп, а саме:

* складання по модулю 216
* множення по модулю 216 + 1
* побітове виключає АБО (XOR).

Математичний опис.

* Блок відкритого тексту розміром 64 біт ділиться на чотири рівні під блоку розміром по 16 біт *(D1(0), D2(0), D3(0), D4(0))*
* Для кожного раунду *(і = 1…8)* вираховується:

*A(i) = D1(i-1) \* K1(i)*

*B(i) = D2(i-1) + K2(i)*

*C(i) = D3(i-1) + K3(i)*

*D(i) = D4(i-1) \* K4(i)*

*E(i) = A(i) C(i)*

*F(i) = B(i) D(i)*

*D1(i) = A(i) ((F(i)+ E(i)\* K5(i))\* K6(i))*

*D2(i) = C(i) ((F(i)+ E(i)\* K5(i))\* K6(i))*

*D3(i) = B(i) (E(i)\* K5(i)+(F(i)+ E(i)\* K5(i))\* K6(i))*

*D4(i) = B(i) (E(i)\* K5(i)+(F(i)+ E(i)\* K5(i))\* K6(i))*

Результатом виконання восьми раундів будуть наступні чотири під блоку *(D1(8), D2(8), D3(8), D4(8))*

* Виконується вихідне перетворення *(і = 9):*

*D1(9) = D1(8) \* K1(9)*

*D2(9) = D3(8) + K2(9)*

*D3(9) = D2(8) \* K3(9)*

*D4(9) = D4(8) \* K4(9)*

Результатом виконання вихідного перетворення є зашифрований текст

*(D1(9), D2(9), D3(9), D4(9))*

Гост28147-89.

Для зашифровування в цьому режимі 64-бітний блок відкритого тексту спочатку розбивається на дві половини (молодші біти - A, старші біти - B). На i-му циклі використовується з'єднання Ki:

*Ai+1 = B f(Ai, Ki)* ( *=* двійкове «виключаючи або»)

*Bi+1 = Ai*

Для генерації під ключів вихідний 256-бітний ключ розбивається на вісім 32-бітових блоків: K1 ... K8.

Ключі K9 ... K24 є циклічним повторенням ключів K1 ... K8 (нумеруються від молодших бітів до старших). Ключі K25 ... K32 є ключами K8 ... K1.

Після виконання всіх 32 раундів алгоритму, блоки A33 і B33 склеюються (зверніть увагу, що старшим блоком стає A33, а молодшим - B33) - результат є результат роботи алгоритму.  
Розшифрування виконується так само [уточнити], як і зашифровування, але інвертується порядок під ключів Ki.

Функція *f(Ai, Ki)* обчислюється таким чином: Ai і Ki складаються по модулю 232. Результат розбивається на вісім 4-бітових підпослідовностей, кожна з яких надходить на вхід свого вузла таблиці замін (в порядку зростання старшинства бітів), званого нижче S-блоком. Загальна кількість S-блоків стандарту - вісім, тобто стільки ж, скільки і підпослідовностей. Кожен S-блок являє собою перестановку чисел від 0 до 15 (конкретний вид S-блоків в стандарті не визначено). Перша 4-бітна підпослідовність потрапляє на вхід першого S-блоку, друга - на вхід другого і т. д.

Якщо вузол S-блоку виглядає так:

1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12

і на вході S-блоку 0, то на виході буде 1, якщо 4, то на виході буде 5, якщо на вході 12, то на виході 6 і т. д.

Виходи всіх восьми S-блоків об'єднуються в 32-бітове слово, потім все слово циклічно зсувається вліво (до старшим розрядам) на 11 бітів.

Режим простої заміни має такі недоліки:

* Може застосовуватися тільки для шифрування відкритих текстів з довжиною, кратною 64 біт
* При шифруванні однакових блоків відкритого тексту виходять однакові блоки шифротекста, що може дати певну інформацію криптоаналітику.

Загальні характеристики.

Операційна система

Windows 7 Ultimate 64-bit SP1

Процесор

Intel Core i7 3610QM @ 2.30GHz

Ivy Bridge 22nm Technology

Оперативна пам'ять

8,00ГБ Dual-Channel DDR3 @ 798MHz

Материнська плата

ASUSTeK COMPUTER INC. G75VW

Відеокарта

2048MB NVDIA GEFORCE GTX 660M

Зберігання

238GB LITEONIT LAT-256M3S ATA Device (SSD)

931GB Seagate ST1000LM024 HN-M101MBB ATA Device (SATA)  
Безкоштовна бібліотека C ++ для криптографічних схем, спочатку написана Wei Dai, і включає в себе шифри, коди автентифікації повідомлень, односторонніх хеш-функцій, криптосистеми публічного ключа, схеми ключових договорів та дефляцію стиснення.

Crypto++ зазвичай забезпечує повну криптографічну реалізацію, і часто включає менш популярні, рідше використовувані схеми. Наприклад, Camellia - блок-шифр, схвалений ISO / NESSIE / IETF, приблизно еквівалентний AES, а Whirlpool - це хеш-функція, схвалена ISO / NESSIE / IETF, приблизно еквівалентна SHA; обидва включені в бібліотеку.

Часові заміри виконуються на мові С++ за допомогою бібліотеки «ctime»

У якій є функція clock(), яка відображає час у мілісекундах.

На мові Python часові заміри виконую за допомогою функції time.time() із бібліотеки time, яка відображає час у секундах, та за допомогою unittest.TestCase яка сумує час роботи усіх тестів та виводить результат у секундах, для неї використовують бібліотеку unittest.

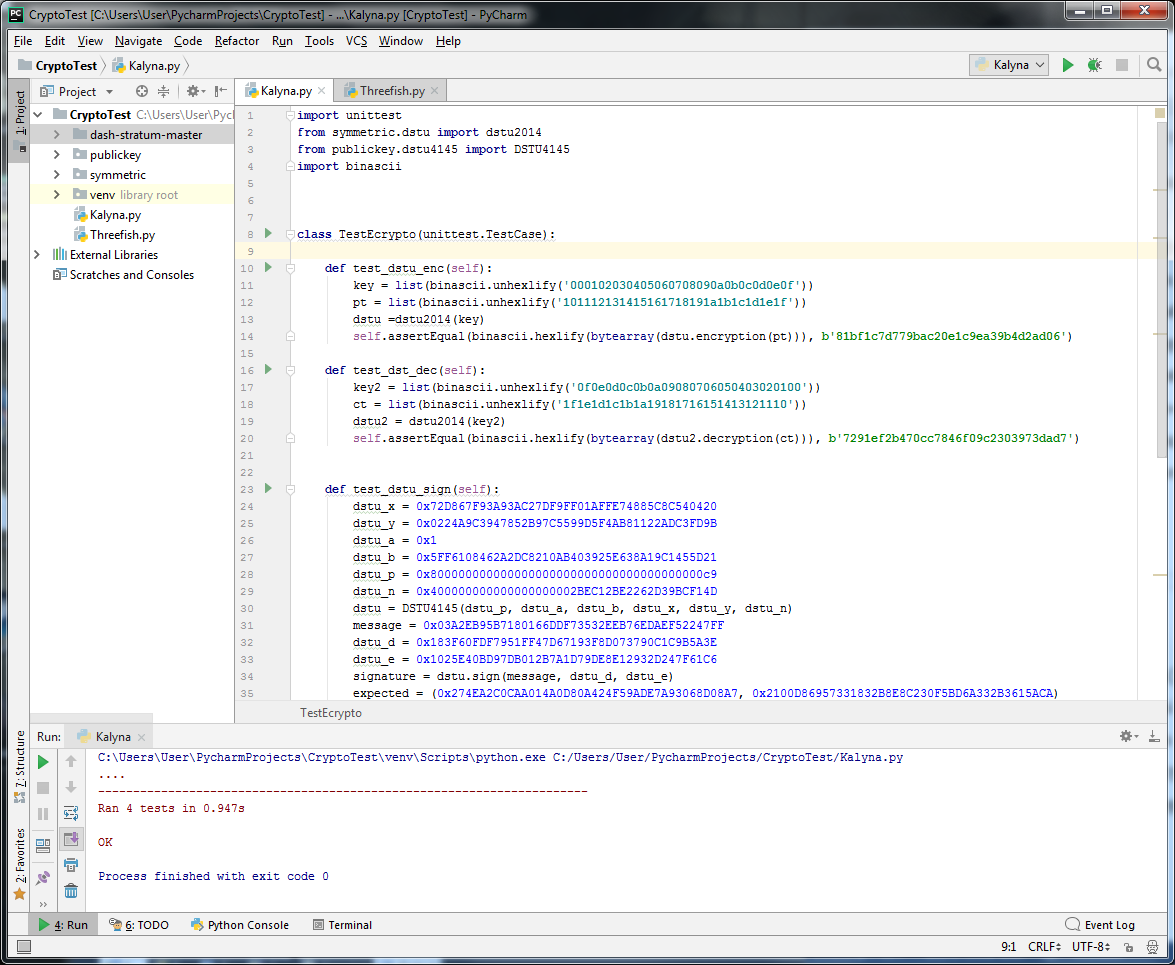


Рис. 1. – Результат роботи блочного шифру «Калина»

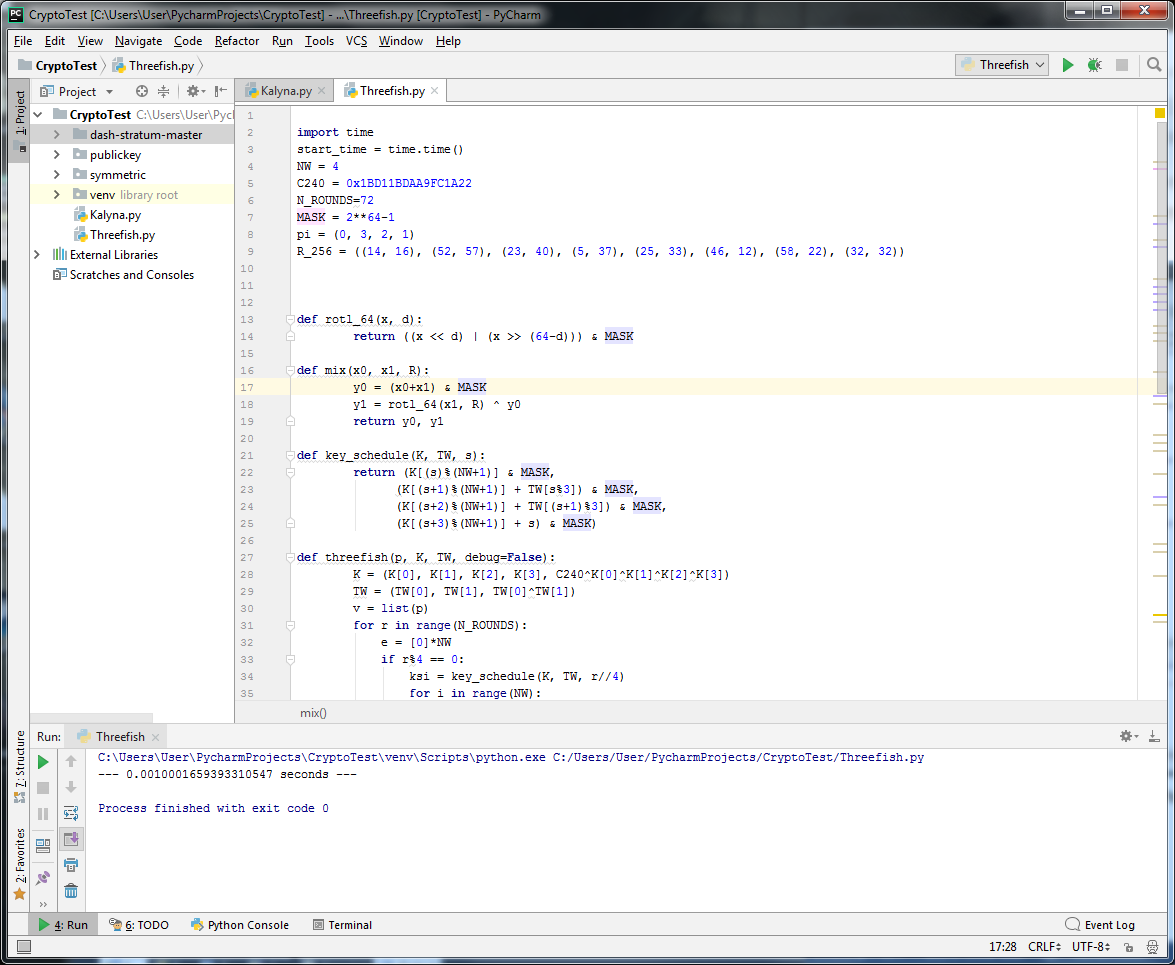


Рис. 2. – Результат роботи блочного шифру «Threefish-256»

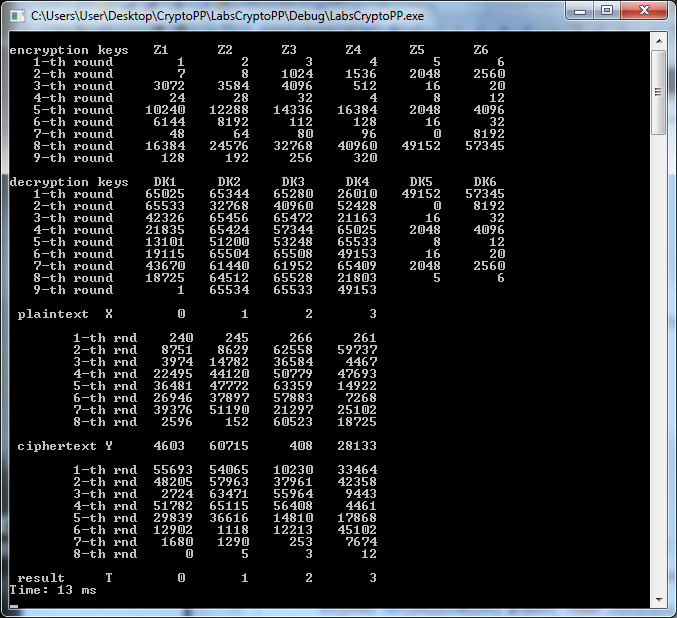


Рис. 3. – Результат роботи блочного шифру «IDEA»

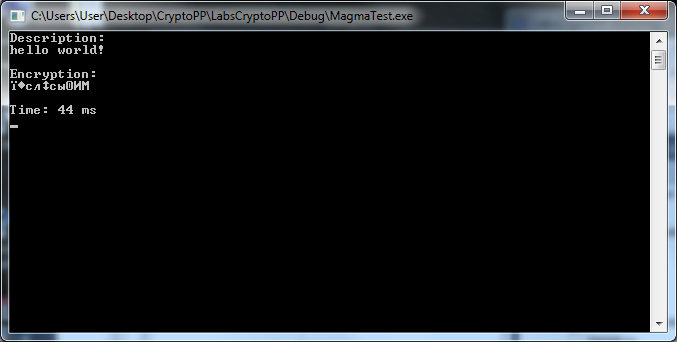


Рис. 4. – Результат роботи блочного шифру «Гост28147-89»

Таблиця результатів замірів часу

|  |  |
| --- | --- |
| Алгоритм шифрування | Час роботи сек. |
| Калина | 0.95 |
| Threefish-256 | 0.001 |
| IDEA | 0.013 |
| Гост28147-89 | 0.044 |

Висновки

Ознайомився з принципами побудови вітчизняного криптографічного стандарту шифрування «Калина». Виконав дослідження його особливостей разом з такими алгоритмами шифрування як «Threefish-256», «IDEA » та «Гост28147-89».

Лістинг програми на Python

Шифр Калина

Kalyna.py

**import** unittest  
**from** symmetric.dstu **import** dstu2014  
**from** publickey.dstu4145 **import** DSTU4145  
**import** binascii  
  
**class** TestEcrypto(unittest.TestCase):  
  
 **def** test\_dstu\_enc(self):  
 key = list(binascii.unhexlify(**'000102030405060708090a0b0c0d0e0f'**))  
 pt = list(binascii.unhexlify(**'101112131415161718191a1b1c1d1e1f'**))  
 dstu =dstu2014(key)  
 self.assertEqual(binascii.hexlify(bytearray(dstu.encryption(pt))), **b'81bf1c7d779bac20e1c9ea39b4d2ad06'**)  
  
 **def** test\_dst\_dec(self):  
 key2 = list(binascii.unhexlify(**'0f0e0d0c0b0a09080706050403020100'**))  
 ct = list(binascii.unhexlify(**'1f1e1d1c1b1a19181716151413121110'**))  
 dstu2 = dstu2014(key2)  
 self.assertEqual(binascii.hexlify(bytearray(dstu2.decryption(ct))), **b'7291ef2b470cc7846f09c2303973dad7'**)  
  
 **def** test\_dstu\_sign(self):  
 dstu\_x = 0x72D867F93A93AC27DF9FF01AFFE74885C8C540420  
 dstu\_y = 0x0224A9C3947852B97C5599D5F4AB81122ADC3FD9B  
 dstu\_a = 0x1  
 dstu\_b = 0x5FF6108462A2DC8210AB403925E638A19C1455D21  
 dstu\_p = 0x800000000000000000000000000000000000000c9  
 dstu\_n = 0x400000000000000000002BEC12BE2262D39BCF14D  
 dstu = DSTU4145(dstu\_p, dstu\_a, dstu\_b, dstu\_x, dstu\_y, dstu\_n)  
 message = 0x03A2EB95B7180166DDF73532EEB76EDAEF52247FF  
 dstu\_d = 0x183F60FDF7951FF47D67193F8D073790C1C9B5A3E  
 dstu\_e = 0x1025E40BD97DB012B7A1D79DE8E12932D247F61C6  
 signature = dstu.sign(message, dstu\_d, dstu\_e)  
 expected = (0x274EA2C0CAA014A0D80A424F59ADE7A93068D08A7, 0x2100D86957331832B8E8C230F5BD6A332B3615ACA)  
 self.assertEqual(signature, expected)  
  
 **def** test\_dstu\_verify(self):  
 dstu\_x = 0x72D867F93A93AC27DF9FF01AFFE74885C8C540420  
 dstu\_y = 0x0224A9C3947852B97C5599D5F4AB81122ADC3FD9B  
 dstu\_a = 0x1  
 dstu\_b = 0x5FF6108462A2DC8210AB403925E638A19C1455D21  
 dstu\_p = 0x800000000000000000000000000000000000000c9  
 dstu\_n = 0x400000000000000000002BEC12BE2262D39BCF14D  
 dstu = DSTU4145(dstu\_p, dstu\_a, dstu\_b, dstu\_x, dstu\_y, dstu\_n)  
 message = 0x03A2EB95B7180166DDF73532EEB76EDAEF52247FF  
 dstu\_d = 0x183F60FDF7951FF47D67193F8D073790C1C9B5A3E  
 dstu\_Q = dstu.gen\_keys(dstu\_d)[1]  
 signature = (0x274EA2C0CAA014A0D80A424F59ADE7A93068D08A7, 0x2100D86957331832B8E8C230F5BD6A332B3615ACA)  
 self.assertEqual(dstu.verify(message, signature, dstu\_Q), **True**)  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

Лістинг програми на Python

Шифр Threefish-256

Threefish.py

**import** time  
start\_time = time.time()  
NW = 4  
C240 = 0x1BD11BDAA9FC1A22  
N\_ROUNDS=72  
MASK = 2\*\*64-1  
pi = (0, 3, 2, 1)  
R\_256 = ((14, 16), (52, 57), (23, 40), (5, 37), (25, 33), (46, 12), (58, 22), (32, 32))  
  
**def** rotl\_64(x, d):  
 **return** ((x << d) | (x >> (64-d))) & MASK  
  
**def** mix(x0, x1, R):  
 y0 = (x0+x1) & MASK  
 y1 = rotl\_64(x1, R) ^ y0  
 **return** y0, y1  
  
**def** key\_schedule(K, TW, s):  
 **return** (K[(s)%(NW+1)] & MASK,  
 (K[(s+1)%(NW+1)] + TW[s%3]) & MASK,  
 (K[(s+2)%(NW+1)] + TW[(s+1)%3]) & MASK,  
 (K[(s+3)%(NW+1)] + s) & MASK)  
  
**def** threefish(p, K, TW, debug=**False**):  
 K = (K[0], K[1], K[2], K[3], C240^K[0]^K[1]^K[2]^K[3])  
 TW = (TW[0], TW[1], TW[0]^TW[1])  
 v = list(p)  
 **for** r **in** range(N\_ROUNDS):  
 e = [0]\*NW  
 **if** r%4 == 0:  
 ksi = key\_schedule(K, TW, r//4)  
 **for** i **in** range(NW):  
 e[i] = (v[i] + ksi[i]) & MASK  
 **if** debug: print(**'key injection '**, list(map(hex, e)))  
 **else**:  
 e = v  
 f = [0]\*NW  
 f[0], f[1] = mix(e[0], e[1], R\_256[r%8][0])  
 f[2], f[3] = mix(e[2], e[3], R\_256[r%8][1])  
 **if** (r%2 == 0) **and** debug: print(**'end of round %03i'** % (r+1), list(map(hex, f)))  
 **for** i **in** range(NW):  
 v[i] = f[pi[i]]  
 **if** (r%2 == 1) **and** debug: print(**'end of round %03i'** % (r+1), list(map(hex, v)))  
  
 ksi = key\_schedule(K, TW, N\_ROUNDS//4)  
 v = [((x + k)^pp) & MASK **for** x, k, pp **in** zip(v, ksi, p)]  
 **if** debug: print(list(map(hex, v)))  
  
 **return** v  
*# Test run with parameters from the NIST CD, file KAT\_MCT/skein\_golden\_kat\_internals.txt*K = (0x0,)\*NW  
TW = (0x0,0x0)  
  
c = threefish((0x0,)\*NW, K, TW)  
*#print([hex(x) for x in c])*K = (0x1716151413121110, 0x1F1E1D1C1B1A1918, 0x2726252423222120, 0x2F2E2D2C2B2A2928)  
TW = (0x0706050403020100, 0x0F0E0D0C0B0A0908)  
  
c = threefish((0xF8F9FAFBFCFDFEFF, 0xF0F1F2F3F4F5F6F7, 0xE8E9EAEBECEDEEEF, 0xE0E1E2E3E4E5E6E7), K, TW)  
*#print([hex(x) for x in c])***if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 print(**"--- %s seconds ---"** % (time.time() - start\_time))

Лістинг програми на C++

Шифр IDEA

Main.cpp

/\* C - program of block cipher IDEA \*/

#include <stdio.h>

#include <conio.h>

#include <iostream>

#include <ctime>

using namespace std;

#define maxim 65537

#define fuyi 65536

#define one 65535

#define round 8

void cip(unsigned IN[5],unsigned OUT[5], unsigned Z[7][10]);

void key(short unsigned uskey[9], unsigned Z[7][10]);

void de\_key(unsigned Z[7][10], unsigned DK[7][10]);

unsigned inv(unsigned xin);

unsigned mul(unsigned a, unsigned b);

void main()

{

unsigned int start\_time = clock();// начальное время

int i,j,x;

unsigned Z[7][10], DK[7][10], XX[5], TT[5], YY[5];

short unsigned uskey[9];

for(i=1;i<=8;i++) uskey[i]=i;

key(uskey,Z); /\* generate encryption subkeys Z[i][r] \*/

printf("\nencryption keys\t Z1\t Z2\t Z3\t Z4\t Z5\t Z6");

for(j=1;j<=9;j++) { printf("\n %3d-th round",j);

if(j==9) for(i=1;i<=4;i++) printf("\t%6d",Z[i][j]);

else for(i=1;i<=6;i++) printf("\t%6d",Z[i][j]);

}

de\_key(Z,DK); /\* compute decryption subkeys DK[i][r] \*/

printf("\n\ndecryption keys\t DK1\t DK2\t DK3\t DK4\t DK5\t DK6");

for(j=1;j<=9;j++) { printf("\n %3d-th round",j);

if(j==9) for(i=1;i<=4;i++) printf("\t%6d",DK[i][j]);

else for(i=1;i<=6;i++) printf("\t%6d",DK[i][j]);

}

for(x=1;x<=4;x++) XX[x]=x-1;

printf("\n\n plaintext X\t%6u\t%6u\t%6u\t%6u\n",

XX[1],XX[2],XX[3],XX[4]);

cip(XX,YY,Z); /\* encipher XX to YY with key Z \*/

printf("\n\n ciphertext Y\t%6u\t%6u\t%6u\t%6u\n",

YY[1],YY[2],YY[3],YY[4]);

cip(YY,TT,DK); /\* decipher YY to TT with key DK \*/

printf("\n\n result T\t%6u\t%6u\t%6u\t%6u\n",

TT[1],TT[2],TT[3],TT[4]);

unsigned int end\_time = clock();// конечное время

unsigned int search\_time = end\_time - start\_time; // искомое время

cout << "Time: " <<search\_time << " ms" << endl;

\_getch();

}

/\* encrypt algorithm \*/

void cip(unsigned IN[5],unsigned OUT[5],unsigned Z[7][10])

{

unsigned r,x1,x2,x3,x4,kk,t1,t2,a;

x1=IN[1]; x2=IN[2]; x3=IN[3]; x4=IN[4];

for(r=1;r<=8;r++) /\* the round function \*/

{

/\* the group operation on 64-bits block \*/

x1 = mul(x1,Z[1][r]); x4 = mul(x4,Z[4][r]);

x2 = (x2 + Z[2][r]) & one; x3 = (x3 + Z[3][r]) & one;

/\* the function of the MA structure \*/

kk = mul(Z[5][r],(x1^x3));

t1 = mul(Z[6][r],(kk+(x2^x4)) & one);

t2 = (kk+t1) & one;

/\* the involutary permutation PI \*/

x1 = x1^t1; x4=x4^t2;

a = x2^t2; x2=x3^t1; x3=a;

printf("\n\t%1u-th rnd %6u\t%6u\t%6u\t%6u",r,x1,x2,x3,x4);

}

/\* the output transformation \*/

OUT[1] = mul(x1,Z[1][round+1]);

OUT[4] = mul(x4,Z[4][round+1]);

OUT[2] = (x3+Z[2][round+1]) & one;

OUT[3] = (x2+Z[3][round+1]) & one;

}

/\* multiplication using the Low-High algorithm \*/

unsigned mul(unsigned a,unsigned b)

{

long int p;

long unsigned q;

if(a==0) p=maxim-b;

else

if(b==0) p=maxim-a;

else {

q=(unsigned long)a\*(unsigned long)b;

p=(q & one) - (q>>16);

if(p<=0) p=p+maxim;

}

return (unsigned)(p&one);

}

/\* compute inverse of xin by Euclidean gcd alg. \*/

unsigned inv(unsigned xin)

{

long n1,n2,q,r,b1,b2,t;

if(xin==0) b2=0;

else

{ n1=maxim; n2=xin; b2=1; b1=0;

do { r = (n1 % n2); q = (n1-r)/n2;

if(r==0) { if(b2<0) b2=maxim+b2; }

else { n1=n2; n2=r; t=b2; b2=b1-q\*b2; b1=t; }

} while (r!=0);

}

return (unsigned)b2;

}

/\* generate encryption subkeys Z's \*/

void key(short unsigned uskey[9], unsigned Z[7][10])

{

short unsigned S[54];

int i,j,r;

for(i=1;i<9;i++) S[i-1]=uskey[i];

/\* shifts \*/

for(i=8;i<54;i++)

{

if((i+2)%8 == 0) /\* for S[14],S[22],... \*/

S[i] = ((S[i-7]<<9) ^ (S[i-14]>>7)) & one;

else if((i+1)%8==0) /\* for S[15],S[23],... \*/

S[i] = ((S[i-15]<<9) ^ (S[i-14]>>7)) & one;

else

S[i] = ((S[i-7]<<9) ^ (S[i-6]>>7)) & one;

}

/\* get subkeys \*/

for(r=1;r<=round+1;r++)

for(j=1;j<7;j++)

Z[j][r]=S[6\*(r-1)+j-1];

}

/\* compute decryption subkeys DK's \*/

void de\_key(unsigned Z[7][10],unsigned DK[7][10])

{

int j;

for(j=1;j<=round+1;j++)

{

DK[1][round-j+2] = inv(Z[1][j]);

DK[4][round-j+2] = inv(Z[4][j]);

if (j==1 || j==round+1) {

DK[2][round-j+2] = (fuyi-Z[2][j]) & one;

DK[3][round-j+2] = (fuyi-Z[3][j]) & one;

} else {

DK[2][round-j+2] = (fuyi-Z[3][j]) & one;

DK[3][round-j+2] = (fuyi-Z[2][j]) & one;

}

}

for(j=1;j<=round+1;j++)

{ DK[5][round+1-j]=Z[5][j];

DK[6][round+1-j]=Z[6][j];

}

}

Лістинг програми на C++

Шифр Гост28147-89

Main.cpp

#include <Windows.h>

//#include <vcl.h>

#include <stdio.h>

#include <conio.h>

#include <clocale>

#include <ctime>

#include <iostream>

#include <fstream>

using namespace std;

#define SIZE 512

//---------------------------------------------------------------------------

// взято из хелпа, определяем размер файла

long filesize(FILE \*stream)

{

long curpos, length;

curpos = ftell(stream);

fseek(stream, 0L, SEEK\_END);

length = ftell(stream);

fseek(stream, curpos, SEEK\_SET);

return length;

}

// функция, реализующая работу ГОСТ 28147-89 в режиме простой замены

void rpz(int rezh, char\* opener, char\* saver)

{

FILE \*f\_begin, \*f\_end; // потоки для исходного и конечного файлов

// таблица замен

byte Tab\_Z[8][16] =

{

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF

};

// ключик

unsigned long key[8] =

{

0x0123,

0x4567,

0x89AB,

0xCDEF,

0x0123,

0x4567,

0x89AB,

0xCDEF

};

char N[4]; // 32-разрядный накопитель,

unsigned long n1=0, n2=0, SUM232=0; // накопители N1, N2, и сумматор

// открываем файлы

f\_begin = fopen (opener,"rb");

f\_end = fopen (saver,"wb");

// определим количество блоков

float blokoff;

blokoff = 8\*filesize(f\_begin);

blokoff = blokoff/64;

int block = blokoff;

if (blokoff-block>0) block++;

int sh;

if (filesize(f\_begin)>=4) sh = 4; else sh = filesize(f\_begin);

int sh1 = 0;

int flag=0;

// начнем считывание и преобразование блоков

// присутствуют проверки на полноту блоков, чтобы считать только нужное количество бит

for (int i=0; i<block; i++)

{

// записываем в накопитель N1

for (int q=0; q<4; q++) \*((byte \*)&N+q) = 0x00;

if ((sh1+sh)<filesize(f\_begin))

{

fread (N,sh,1,f\_begin);

sh1+=sh;

}

else

{

sh=filesize(f\_begin)-sh1;

fread (N,sh,1,f\_begin);

flag=1;

}

n1 = \*((unsigned long \*)&N);

// записываем в накопитель N2

for (int q=0; q<4; q++) \*((byte \*)&N+q) = 0x00;

if ((sh1+sh)<filesize(f\_begin))

{

fread (N,sh,1,f\_begin);

sh1+=sh;

}

else

{

if (flag==0)

{

sh=filesize(f\_begin)-sh1;

fread (N,sh,1,f\_begin);

}

}

n2 = \*((unsigned long \*)&N);

// 32 цикла простой замены

// ключ считываем в требуемом ГОСТом порядке

int c = 0;

for (int k=0; k<32; k++)

{

if (rezh==1) { if (k==24) c = 7; }

else { if (k==8) c = 7; }

// суммируем в сумматоре СМ1

SUM232 = key[c] + n1;

// заменяем по таблице замен

byte first\_byte=0,second\_byte=0,zam\_symbol=0;

int n = 7;

for (int q=3; q>=0; q--)

{

zam\_symbol = \*((byte \*)&SUM232+q);

first\_byte = (zam\_symbol & 0xF0) >> 4;

second\_byte = (zam\_symbol & 0x0F);

first\_byte = Tab\_Z[n][first\_byte];

n--;

second\_byte = Tab\_Z[n][second\_byte];

n--;

zam\_symbol = (first\_byte << 4) | second\_byte;

\*((byte \*)&SUM232+q) = zam\_symbol;

}

SUM232 = (SUM232<<11)|(SUM232>>21); // циклический сдвиг на 11

SUM232 = n2^SUM232; // складываем в сумматоре СМ2

if (k<31)

{

n2 = n1;

n1 = SUM232;

}

if (rezh==1)

{

if (k<24)

{

c++;

if (c>7) c = 0;

}

else

{

c--;

if (c<0) c = 7;

}

}

else

{

if (k<8)

{

c++;

if (c>7) c = 0;

}

else

{

c--;

if (c<0) c = 7;

}

}

}

n2 = SUM232;

// вывод результата в файл

char sym\_rez;

for (int q=0; q<=3; q++)

{

sym\_rez = \*((byte \*)&n1+q);

fprintf(f\_end, "%c", sym\_rez);

}

for (int q=0; q<=3; q++)

{

sym\_rez = \*((byte \*)&n2+q);

fprintf(f\_end, "%c", sym\_rez);

}

}

fclose (f\_begin);

fclose (f\_end);

}

//---------------------------------------------------------------------------

int main()

{

unsigned int start\_time = clock();// начальное время

setlocale(LC\_ALL, "Rus");

// выбираем шифрование или расшифрование

char \*vec1 = new char[SIZE], \*vec2 = new char[SIZE], \*vec3 = new char[SIZE];

FILE \* ptrFile = fopen("D:\\test.txt" , "r");

fgets(vec1, 100, ptrFile);

cout << "Description: \n";

puts(vec1);

fclose (ptrFile);

FILE \* ptrFile1 = fopen("D:\\test1.txt" , "r");

fgets(vec2, 100, ptrFile1);

cout << endl;

printf("Encryption: \n");

puts(vec2);

fclose (ptrFile1);

rpz(rezhim, "D:\\test.txt","D:\\test1.txt"); // запускаем РПЗ

unsigned int end\_time = clock();// конечное время

unsigned int search\_time = end\_time - start\_time; // искомое время

cout << endl;

cout << "Time: " <<search\_time << " ms" << endl;

\_getch();

return 0;

}