Звіт до лабораторної роботи з криптографії

*“Алгоритм шифрування Blowfish”*

Пядика Любомира, МІ-3, 2018 р.

# Постановка задачі

Потрібно було реалізувати алгоритм Blowfish - криптографічний алгоритм, який реалізує блочне симетричне шифрування.

Розроблений Брюсом Шнайєром в 1993 році. Являє собою шифр на основі мережі Фейстеля. Виконано на простих і швидких операціях: XOR, підстановка, додавання. Не запатентований і вільно поширюваний.

# Опис алгоритму

**Параметри**

Секретний ключ K (від 32 до 448 біт)

32-бітові ключі шифрування P1-P18

32-бітові таблиці замін S1-S4:

S1 [0] S1 [1] .. S1 [255]

S2 [0] S2 [1] .. S2 [255]

S3 [0] S3 [1] .. S3 [255]

S4 [0] S4 [1] .. S4 [255]

**Функція F (x)**

32-бітний блок ділиться на чотири 8-бітних блоки (X1, X2, X3, X4), кожен з яких є індексом масиву таблиці замін S1-S4

Значення S1[X1] і S2[X2] складаються по модулю 2^32, після "XOR"яться з S3[X3] і, нарешті, додаються з S4[X4] по модулю 2^32.

Результат цих операцій — значення F (x).

**Етапи алгоритму**

Розділений на 2 етапи:

Підготовчий — формування ключів шифрування по секретному ключу.

Ініціалізація масивів P і S за допомогою секретного ключа K

Ініціалізація P1-P18 фіксованим рядком, що складається з шістнадцяткових цифр мантиси числа пі.

Проводиться операція XOR над P1 з першими 32 бітами ключа K, над P2 з другими 32-бітами і так далі.

Якщо ключ K коротше, то він накладається циклічно.

Шифрування ключів і таблиць замін

Алгоритм шифрування 64-бітного блоку, використовуючи початкові ключі P1-P18 і таблицю замін S1-S4, шифрує 64 бітну нульовий (0x0000000000000000) рядок. Результат записується в P1, P2.

P1 і P2 шифруються зміненими значеннями ключів і таблиць замін. Результат записується в P3 і P4.

Шифрування триває до зміни всіх ключів P1-P18 і таблиць замін S1-S4.

Шифрування тексту отриманими ключами і F(x), з попереднім розбиттям на блоки по 64 біти. Якщо неможливо розбити початковий текст точно на блоки по 64 біти, використовуються різні режими шифрування для побудови повідомлення, що складається з цілого числа блоків. Сумарні необхідна пам'ять 4168 байт: P1-P18: 18 змінних по 32 біта; S1-S4: 4x256 змінних по 32 бита.

Розшифрування відбувається аналогічно, тільки P1-P18 застосовуються у зворотному порядку.

**Вибір початкового значення P-масиву і таблиці замін**

Немає нічого особливого в цифрах числа пі. Цей вибір полягає в ініціалізації послідовності, не пов'язаної з алгоритмом, яка могла б бути збережена як частина алгоритму або отримана при необхідності (Пі (число)). Як вказує Брюс Шнайєр: «Підійде будь-який рядок з випадкових бітів цифр числа e, RAND-таблиці, або випадкові згенеровані цифри.»

# Програмна реалізація

#define S(x, i) (SBoxes[i][x.w.byte##i])

#define bf\_F(x) (((S(x, 0) + S(x, 1)) ^ S(x, 2)) + S(x, 3))

#define ROUND(a, b, n) (a.dword ^= bf\_F(b) ^ PArray[n])

CBlowFish::CBlowFish()

{

PArray = new DWORD[18];

SBoxes = new DWORD[4][256];

}

CBlowFish::~CBlowFish()

{

delete PArray;

delete[] SBoxes;

}

void CBlowFish::Blowfish\_encipher(DWORD\* xl, DWORD\* xr)

{

union aword Xl, Xr;

Xl.dword = \*xl;

Xr.dword = \*xr;

Xl.dword ^= PArray[0];

ROUND(Xr, Xl, 1); ROUND(Xl, Xr, 2);

ROUND(Xr, Xl, 3); ROUND(Xl, Xr, 4);

ROUND(Xr, Xl, 5); ROUND(Xl, Xr, 6);

ROUND(Xr, Xl, 7); ROUND(Xl, Xr, 8);

ROUND(Xr, Xl, 9); ROUND(Xl, Xr, 10);

ROUND(Xr, Xl, 11); ROUND(Xl, Xr, 12);

ROUND(Xr, Xl, 13); ROUND(Xl, Xr, 14);

ROUND(Xr, Xl, 15); ROUND(Xl, Xr, 16);

Xr.dword ^= PArray[17];

\*xr = Xl.dword;

\*xl = Xr.dword;

}

void CBlowFish::Blowfish\_decipher(DWORD\* xl, DWORD\* xr)

{

union aword Xl;

union aword Xr;

Xl.dword = \*xl;

Xr.dword = \*xr;

Xl.dword ^= PArray[17];

ROUND(Xr, Xl, 16); ROUND(Xl, Xr, 15);

ROUND(Xr, Xl, 14); ROUND(Xl, Xr, 13);

ROUND(Xr, Xl, 12); ROUND(Xl, Xr, 11);

ROUND(Xr, Xl, 10); ROUND(Xl, Xr, 9);

ROUND(Xr, Xl, 8); ROUND(Xl, Xr, 7);

ROUND(Xr, Xl, 6); ROUND(Xl, Xr, 5);

ROUND(Xr, Xl, 4); ROUND(Xl, Xr, 3);

ROUND(Xr, Xl, 2); ROUND(Xl, Xr, 1);

Xr.dword ^= PArray[0];

\*xl = Xr.dword;

\*xr = Xl.dword;

}

void CBlowFish::Initialize(BYTE key[], int keybytes)

{

int i, j;

DWORD data, datal, datar;

union aword temp;

for (i = 0; i < 18; i++)

PArray[i] = bf\_P[i];

for (i = 0; i < 4; i++)

{

for (j = 0; j < 256; j++)

SBoxes[i][j] = bf\_S[i][j];

}

j = 0;

for (i = 0; i < NPASS + 2; ++i)

{

temp.dword = 0;

temp.w.byte0 = key[j];

temp.w.byte1 = key[(j + 1) % keybytes];

temp.w.byte2 = key[(j + 2) % keybytes];

temp.w.byte3 = key[(j + 3) % keybytes];

data = temp.dword;

PArray[i] ^= data;

j = (j + 4) % keybytes;

}

datal = 0;

datar = 0;

for (i = 0; i < NPASS + 2; i += 2)

{

Blowfish\_encipher(&datal, &datar);

PArray[i] = datal;

PArray[i + 1] = datar;

}

for (i = 0; i < 4; ++i)

{

for (j = 0; j < 256; j += 2)

{

Blowfish\_encipher(&datal, &datar);

SBoxes[i][j] = datal;

SBoxes[i][j + 1] = datar;

}

}

}

DWORD CBlowFish::GetOutputLength(DWORD lInputLong)

{

DWORD lVal;

lVal = lInputLong % 8;

if (lVal != 0)

return lInputLong + 8 - lVal;

else

return lInputLong;

}

DWORD CBlowFish::Encode(BYTE\* pInput, BYTE\* pOutput, DWORD lSize)

{

DWORD lOutSize = GetOutputLength(lSize);

for (DWORD lCount = 0; lCount < lOutSize; lCount += 8)

{

if (lCount < lSize - 7)

{

BYTE\* pi = pInput + lCount;

BYTE\* po = pOutput + lCount;

for (int i = 0; i < 8; i++)

\*po++ = \*pi++;

Blowfish\_encipher((DWORD\*)(pOutput + lCount), (DWORD\*)(pOutput + lCount + 4));

}

else

{

DWORD lGoodBytes = lSize - lCount;

BYTE\* pi = pInput + lCount;

BYTE\* po = pOutput + lCount;

int i;

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

\*po++ = \*pi++;

for (int j = i; j < 8; j++)

\*po++ = 0;

Blowfish\_encipher((DWORD\*)(pOutput + lCount), (DWORD\*)(pOutput + lCount + 4));

}

}

return lOutSize;

}

void CBlowFish::Decode(BYTE\* pInput, BYTE\* pOutput, DWORD lSize)

{

for (DWORD lCount = 0; lCount < lSize; lCount += 8)

{

BYTE\* pi = pInput + lCount;

BYTE\* po = pOutput + lCount;

for (int i = 0; i < 8; i++)

\*po++ = \*pi++;

Blowfish\_decipher((DWORD\*)(pOutput + lCount), (DWORD\*)(pOutput + lCount + 4));

}

}