Министерство образования и науки Российской Федерации

Санкт-Петербургский государственный политехнический университет

—

Факультет технической кибернетики

**Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9**

**Анализ шифра методом 2-адического продолжения многочленов Жегалкина**

по дисциплине «Теория итерированных шифров»

Выполнил

студент гр. 5088/1 И.И. Сиваков

Проверил

ст. преподаватель О.В. Шемякина

Санкт-Петербург

2011

# задание

1. Составить программу, моделирующую шифр. Выбрать произвольный ключ и открытый текст, вычислить шифртекст.
2. Для некоторого множества случайных ключей найти шифртексты и с их помощью найти матрицы (Nij) и оценки преобладания для каждого бита ключа

# Результаты работы

Любая булева функция может быть единственным образом представлена многочленом Жегалкина. Таблично заданную булеву функцию можно записать многочленом Жегалкина путем умножения рекуррентно построенной матрицы *L4* на столбец значений булевой функции.

Продолжение операции сложения по модулю 2 и умножения:

* *x + y -> |x – y| (mod 2m);*
* *xy (mod 2)-> xy (mod 2m).*

Такое продолжение операции сложения по модулю 2 нарушает ассоциативность операции, поэтому при использовании указанного продолжения следует по возможности сократить число сложений по модулю 2 и использовать обратные элементы и операцию умножения.

2-адическое показательное нормирование *val*: для *n = 2lc*, где с – нечетное число, *val(n) = -l*. Так как *val(1) = 0, val(0) = -m, val(2) = -1*, то значение 2 является промежуточным между 0 и 1.

Целевая функция шифра представляет собой продолженную конъюнкцию поразрядных равенств промежуточных текстов, полученных в результате шифрования и дешифрования на 2 циклах. В связи с введенным продолжением многочленов Жегалкина у целевой функции появляются локальные экстремумы, не соответствующие решению задачи криптоанализа.

Размер блока исследуемого шифра 32 бита. Зашифрование выполняется на четырех циклах. На каждом цикле используется один и тот же ключ, выполняются действия:

1. XOR блока текста с блоком ключа.
2. Подстановка S блоков по 4 бита.
3. 32-х битовая перестановка P.

После выполнения 4-х циклов выполняется XOR блока текста с блоком ключа.

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

P:

P: (11, 16, 21, 26, 31, 4, 9, 14, 19, 24, 29, 2, 7, 12, 17, 22, 27, 0, 5, 10, 15, 20, 25, 30, 3, 8, 13, 18, 23, 28, 1, 6)

Прямая подстановка:

v1 = u4 + u3 + u2 + u1\*u3\*u4 + u1\*u2 + u1\*u2\*u3

v2 = u3 + u3\*u4 + u2\*u4 + u1\*u4 + u1\*u3 + u1\*u3\*u4 + u1\*u2 + u1\*u2\*u4 + u1\*u2\*u3

v3 = 1 + u3 + u2 + u2\*u4 + u2\*u3 + u1 + u1\*u3 + u1\*u3\*u4 + u1\*u2\*u4 + u1\*u2\*u3

v4 = u4 + u3 + u2 + u2\*u4 + u1

Обратная подстановка:

u1 = v4 + v2\*v3 + v1

u2 = v3\*v4 + v2 + v2\*v3\*v4 + v1\*v4 + v1\*v3 + v1\*v3\*v4 + v1\*v2 + v1\*v2\*v4

u3 = 1 + v4 + v3 + v2\*v4 + v1\*v3\*v4 + v1\*v2 + v1\*v2\*v4

u3 = 1 + v4 + v3 + v2\*v4 + v1\*v3\*v4 + v1\*v2 + v1\*v2\*v4

Шифрование задано уравнением *y = F4(x) ⊕ k*, где *F4* – шифрование на 4 циклах. Если выполнить 2 цикла дешифрования, то справедливо равенство *F2(x) = F-2(y ⊕ k)*.

Определим целевую функцию:

Для поиска максимума целевой функции используется алгоритм 1.

Выбор числа *m* осуществляется экспериментально. Для рассмотренного примера *m = 27*.

Для каждого цикла строится матрица p2×2. Элемент матрицы pij – частота вскрытия бита аргумента шифра как i, если в действительности он равен j. Строится матрица q2×2. Элемент матрицы qij – условная вероятность того, что истинное значение бита аргумента шифра равно i, тогда как его оценка равна j;

Для некоторых бит получилось большое преобладание вероятностей:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Разряд | q00 | q10 | q01 | q11 |
| 3 | 0.64 | 0.36 | 0.57 | 0.43 |
| 15 | 0.53 | 0.47 | 0.36 | 0.64 |
| 28 | 0.63 | 0.37 | 0.62 | 0.38 |

# Выводы

Рассмотренное продолжение многочленов Жегалкина из-за нарушения ассоциативности влияет на вероятность работы метода.

Шифр является l-эргодическим для 22, 23, 25, 30 и 31 разрядов. Для них при тестировании на 1000 различных ключах и одинаковых начальных приближениях вероятность ошибки в определении знака преобладания была меньше 1/32. Эти разряды можно вскрыть с помощью данного метода.

***Приложение***

//lab7.cpp

#include "stdafx.h"

#define ROUND(x) (x>0.5?1:0)

double V1(double u1, double u2, double u3, double u4)

{

double res;

res =

u1\*u2\*(1.0-u3)\*u4 +

u1\*(1.0-u2)\*u3\*u4 +

u1\*(1.0-u2)\*u3\*(1.0-u4) +

u1\*(1.0-u2)\*(1.0-u3)\*u4 +

(1.0-u1)\*(1.0-u2)\*u3\*(1.0-u4) +

(1.0-u1)\*(1.0-u2)\*(1.0-u3)\*u4 +

(1.0-u1)\*u2\*(1.0-u3)\*(1.0-u4) +

(1.0-u1)\*u2\*u3\*u4;

return res;

}

double V2(double u1, double u2, double u3, double u4)

{

double res;

res =

u1\*u2\*u3\*u4 +

u1\*u2\*(1.0-u3)\*(1.0-u4) +

u1\*(1.0-u2)\*u3\*u4 +

u1\*(1.0-u2)\*(1.0-u3)\*u4 +

(1.0-u1)\*(1.0-u2)\*u3\*(1.0-u4) +

(1.0-u1)\*u2\*u3\*(1.0-u4) +

(1.0-u1)\*u2\*u3\*u4 +

(1.0-u1)\*u2\*(1.0-u3)\*u4;

return res;

}

double V3(double u1, double u2, double u3, double u4)

{

double res;

res =

u1\*u2\*u3\*(1.0-u4) +

u1\*u2\*(1.0-u3)\*u4 +

u1\*u2\*(1.0-u3)\*(1.0-u4) +

u1\*(1.0-u2)\*u3\*u4 +

(1.0-u1)\*u2\*u3\*u4 +

(1.0-u1)\*u2\*(1.0-u3)\*u4 +

(1.0-u1)\*(1.0-u2)\*(1.0-u3)\*u4 +

(1.0-u1)\*(1.0-u2)\*(1.0-u3)\*(1.0-u4);

return res;

}

double V4(double u1, double u2, double u3, double u4)

{

double res;

res =

u1\*u2\*u3\*u4 +

u1\*u2\*u3\*(1.0-u4) +

u1\*(1.0-u2)\*u3\*u4 +

u1\*(1.0-u2)\*(1.0-u3)\*(1.0-u4) +

(1.0-u1)\*(1.0-u2)\*u3\*(1.0-u4) +

(1.0-u1)\*u2\*(1.0-u3)\*u4 +

(1.0-u1)\*u2\*(1.0-u3)\*(1.0-u4) +

(1.0-u1)\*(1.0-u2)\*(1.0-u3)\*u4;

return res;

}

double U1(double v1, double v2, double v3, double v4)

{

double res;

res =

v1\*v2\*(1.0-v3)\*(1.0-v4) +

v1\*(1.0-v2)\*(1.0-v3)\*(1.0-v4) +

v1\*(1.0-v2)\*v3\*(1.0-v4) +

v1\*v2\*v3\*v4 +

(1.0-v1)\*(1.0-v2)\*v3\*v4 +

(1.0-v1)\*(1.0-v2)\*(1.0-v3)\*v4 +

(1.0-v1)\*v2\*(1.0-v3)\*v4 +

(1.0-v1)\*v2\*v3\*(1.0-v4);

return res;

}

double U2(double v1, double v2, double v3, double v4)

{

double res;

res =

(1.0-v1)\*v2\*v3\*v4 +

(1.0-v1)\*v2\*v3\*(1.0-v4) +

(1.0-v1)\*v2\*(1.0-v3)\*v4 +

(1.0-v1)\*v2\*(1.0-v3)\*(1.0-v4) +

v1\*v2\*v3\*(1.0-v4) +

v1\*(1.0-v2)\*v3\*(1.0-v4) +

(1.0-v1)\*(1.0-v2)\*v3\*v4 +

v1\*(1.0-v2)\*(1.0-v3)\*v4;

return res;

}

double U3(double v1, double v2, double v3, double v4)

{

double res;

res =

v1\*v2\*v3\*v4 +

v1\*v2\*v3\*(1.0-v4) +

v1\*v2\*(1.0-v3)\*v4 +

v1\*(1.0-v2)\*(1.0-v3)\*(1.0-v4) +

(1.0-v1)\*v2\*(1.0-v3)\*v4 +

(1.0-v1)\*v2\*(1.0-v3)\*(1.0-v4) +

(1.0-v1)\*(1.0-v2)\*v3\*v4 +

(1.0-v1)\*(1.0-v2)\*(1.0-v3)\*(1.0-v4);

return res;

}

double U4(double v1, double v2, double v3, double v4)

{

double res;

res =

v1\*v2\*v3\*v4 +

v1\*v2\*v3\*(1.0-v4) +

v1\*(1.0-v2)\*v3\*v4 +

v1\*(1.0-v2)\*v3\*(1.0-v4) +

v1\*v2\*(1.0-v3)\*(1.0-v4) +

(1.0-v1)\*v2\*v3\*v4 +

(1.0-v1)\*v2\*(1.0-v3)\*v4 +

(1.0-v1)\*(1.0-v2)\*(1.0-v3)\*(1.0-v4);

return res;

}

void Sub(double \*u)

{

double temp1 = V1(u[0],u[1],u[2],u[3]);

double temp2 = V2(u[0],u[1],u[2],u[3]);

double temp3 = V3(u[0],u[1],u[2],u[3]);

double temp4 = V4(u[0],u[1],u[2],u[3]);

u[0] = temp1;

u[1] = temp2;

u[2] = temp3;

u[3] = temp4;

}

void SubInv(double \*v)

{

double temp1 = U1(v[0],v[1],v[2],v[3]);

double temp2 = U2(v[0],v[1],v[2],v[3]);

double temp3 = U3(v[0],v[1],v[2],v[3]);

double temp4 = U4(v[0],v[1],v[2],v[3]);

v[0] = temp1;

v[1] = temp2;

v[2] = temp3;

v[3] = temp4;

}

double Xor(double x, double k)

{

return x\*(1.0-k) + (1.0-x)\*k;;

}

void Xor(double \*x, double \*k, int size)

{

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

x[i] = Xor(x[i],k[i]);

}

void dtouc(double \*dX, unsigned char \*ucX)

{

unsigned char ucTemp;

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

{

ucX[i] = 0;

ucX[i] += ROUND(dX[i\*4 + 0]);

ucX[i] += ROUND(dX[i\*4 + 1])\*2;

ucX[i] += ROUND(dX[i\*4 + 2])\*2\*2;

ucX[i] += ROUND(dX[i\*4 + 3])\*2\*2\*2;

ucX[i] += ROUND(dX[i\*4 + 4])\*2\*2\*2\*2;

ucX[i] += ROUND(dX[i\*4 + 5])\*2\*2\*2\*2\*2;

ucX[i] += ROUND(dX[i\*4 + 6])\*2\*2\*2\*2\*2\*2;

ucX[i] += ROUND(dX[i\*4 + 7])\*2\*2\*2\*2\*2\*2\*2;

}

}

void uctod(unsigned char \*ucX, double \*dX)

{

unsigned char ucTemp;

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

{

ucTemp = ucX[i];

dX[8\*i+7] = ucTemp%2;

dX[8\*i+6] = ucTemp/2%2;

dX[8\*i+5] = ucTemp/2/2%2;

dX[8\*i+4] = ucTemp/2/2/2%2;

dX[8\*i+3] = ucTemp/2/2/2/2%2;

dX[8\*i+2] = ucTemp/2/2/2/2/2%2;

dX[8\*i+1] = ucTemp/2/2/2/2/2/2%2;

dX[8\*i+0] = ucTemp/2/2/2/2/2/2/2%2;

}

}

double H(double \*dK, double \*dX, double \*dY, Cipher \*ciph)

{

double dF2X[32];

double dFinv2X[32];

double dTemp[32];

double dH = 1.0;

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

dF2X[i] = dX[i];

for(int c = 0; c < 2; c++)

{

Xor(dF2X, dK, 32);

for(int i = 0; i < 32; i+=4)

Sub(dF2X+i);

/\*\*/

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

dTemp[i] = dF2X[i];

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

dF2X[ciph->per->pers[i]] = dTemp[i];

/\*\*/

}

/\*\*/

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

dFinv2X[i] = dY[i];

Xor(dFinv2X, dK, 32);

for(int c = 0; c < 2; c++)

{

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

dTemp[i] = dFinv2X[i];

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

dFinv2X[ciph->per->pers\_inv[i]] = dTemp[i];

for(int i = 0; i < 32; i+=4)

SubInv(dFinv2X+i);

Xor(dFinv2X, dK, 32);

}

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

dH \*= dF2X[i]\*dFinv2X[i] + (1.0 - dF2X[i])\*(1.0 - dFinv2X[i]);

return dH;

}

int main()

{

srand (time(NULL));

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

vector<int> my\_s, my\_p;

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

int my\_sub[16] = {2, 11, 13, 0, 9, 7, 4, 14, 1, 12, 8, 15, 6, 10, 3, 5};

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

my\_s = vector<int>(my\_sub, my\_sub+16);

my\_p = vector<int>(32);

for (int i = 0; i < 32; i++) my\_p[i] = (i\*5 + 11)%32;

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

Cipher ciph(my\_s,my\_p);

cout << "Working.." << endl;

double dKK[32];

double dK[32];

double dX[32];

double dY[32];

double dH[2];

double dHx;

int j = 0;

int r = 17;

int iOpenBits[32];

unsigned char ucX[5] = "\xf5\x6f\x3e\x65";//"0000";

unsigned char ucY[4];

unsigned char ucK[5] = "\x73\x56\xa3\x64";

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

ucY[i] = ucX[i];

ciph.CryptBlock(ucY,ucK);

uctod(ucK, dKK);

uctod(ucX, dX);

uctod(ucY, dY);

cout << "-----------------K orig--------------" << endl;

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

{

cout << dKK[i] << "\t";

if(!((i+1)%8)) cout << endl;

}

cout << endl << endl;

LOOP:

dHx = pow(2.0, -32);

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

dK[i] = 0.5;

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

{

iOpenBits[k] = rand()%32;

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

{

if (iOpenBits[i] == iOpenBits[k]) {k--; break;}

}

}

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

dK[iOpenBits[i]] = dKK[iOpenBits[i]];

j = 0;

while (j < 32)

{

if(dK[j] == 0.5)

{

for(int k = 0; k < 2; k++)

{

if(k == 0) dK[j] = 0.0;

if(k == 1) dK[j] = 1.0;

dH[k] = H(dK, dX, dY, &ciph);

}

if ((dH[0] < dHx)&&(dHx < dH[1])) {dHx = dH[1]; dK[j] = 1;}

else if ((dH[1] < dHx)&&(dHx < dH[0])) {dHx = dH[0]; dK[j] = 0;}

else {goto LOOP;}

}

j++;

}

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

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

if (dK[i] != dKK[i])

{

cout << "key not found" << endl;

cout << "----------opened-K-------------------" << endl;

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

{

cout << dK[i] << "\t";

if(!((i+1)%8)) cout << endl;

}

cout << endl << endl;

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

dK[i] = 0.5;

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

dK[iOpenBits[i]] = dKK[iOpenBits[i]];

cout << "---------initial-K-------------------" << endl;

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

{

cout << dK[i] << "\t";

if(!((i+1)%8)) cout << endl;

}

cout << endl << endl;

cout << "H\* = " << dHx << endl;

cout << "r = " << r << endl;

\_getch();

return 0;

break;

}

cout << "key found" << endl;

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

dK[i] = 0.5;

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

dK[iOpenBits[i]] = dKK[iOpenBits[i]];

cout << "---------initial-K-------------------" << endl;

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

{

cout << dK[i] << "\t";

if(!((i+1)%8)) cout << endl;

}

cout << endl << endl;

cout << "H\* = " << dHx << endl;

cout << "r = " << r << endl;

\_getch();

return 0;

}