Министерство образования и науки Российской Федерации

Санкт-Петербургский государственный политехнический университет

—

Факультет технической кибернетики

**Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №8**

**Анализ шифра методом рационального 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|;*
* *x y -> xy.*

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.

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 = abs(abs(u2\*abs(1 - u1\*abs(1 - u3)) - u3\*abs(1 - u1\*u4)) - u4);

v2 = abs(abs(u3\*abs(1 - u4) - u4\*abs(u2 - u1)) - u1\*abs(abs(1 - u4)\*abs(u3 - u2) - u2\*u3));

v3 = abs(abs(abs(1 - u3) - u2\*abs(1 - u4)) - abs(u1\*abs(u2\*abs(u4 - u3) - u3\*abs(1 - u4)) - abs(u2\*u3 - u1)));

v4= abs abs(abs(u4 - u3) - abs(u2\*abs(1 - u4) - u1));

Обратная подстановка:

u1 = abs(abs(v4 - v2\*v3) - v1);

u2 = abs(abs(v3\*v4\*abs(1 - v2) - abs(v2 - v1\*v4)) - v1\*abs(1 - v4)\*abs(v3 - v2));

u3 = abs(abs(abs(1 - v3) - v4\*abs(1 - v2)) - v1\*abs(v2\*abs(1 - v4) - v3\*v4));

u4 = abs(abs(abs(1 - v4)\*abs(1 - v3) - abs(v2\*abs(1 - v3) - abs(v2\*v3\*v4 - v1))) - v1\*v4\*abs(abs(1 - v3) - v2));

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.

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 |
| 0 | 0.59 | 0.41 | 0.64 | 0.36 |
| 21 | 0.35 | 0.65 | 0.36 | 0.64 |
| 23 | 0.36 | 0.64 | 0.48 | 0.52 |

# Выводы

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

Шифр является l-эргодическим для 2, 3, 6, 11, 13, 15, 17, 19, 24 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***

//lab8.cpp

#include "stdafx.h"

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

{

double res;

res = abs(abs(u2\*abs(1 - u1\*abs(1 - u3)) - u3\*abs(1 - u1\*u4)) - u4);

return res;

}

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

{

double res;

res = abs(abs(u3\*abs(1 - u4) - u4\*abs(u2 - u1)) - u1\*abs(abs(1 - u4)\*abs(u3 - u2) - u2\*u3));

return res;

}

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

{

double res;

res = abs(abs(abs(1 - u3) - u2\*abs(1 - u4)) - abs(u1\*abs(u2\*abs(u4 - u3) - u3\*abs(1 - u4)) - abs(u2\*u3 - u1)));

return res;

}

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

{

double res;

res = abs(abs(u4 - u3) - abs(u2\*abs(1 - u4) - u1));

return res;

}

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

{

double res;

res = abs(abs(v4 - v2\*v3) - v1);

return res;

}

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

{

double res;

res = abs(abs(v3\*v4\*abs(1 - v2) - abs(v2 - v1\*v4)) - v1\*abs(1 - v4)\*abs(v3 - v2));

return res;

}

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

{

double res;

res = abs(abs(abs(1 - v3) - v4\*abs(1 - v2)) - v1\*abs(v2\*abs(1 - v4) - v3\*v4));

return res;

}

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

{

double res;

res = abs(abs(abs(1 - v4)\*abs(1 - v3) - abs(v2\*abs(1 - v3) - abs(v2\*v3\*v4 - v1))) - v1\*v4\*abs(abs(1 - v3) - v2));

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 abs(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 \*= 1 - abs(dF2X[i] - dFinv2X[i]);

return dH;

}

double alg1(double \*dK, double \*dX, double \*dY, Cipher \*ciph)

{

double dH[32];

double dJ[32];

double dHx, dHmax;

int imax;

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

dJ[i] = 0;

dHx = H(dK, dX, dY, ciph);

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

STEP2:

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

{

if(dK[i] == 0.5)

{

double dH0, dH1;

dK[i] = 0.0;

dH0 = H(dK, dX, dY, ciph);

dK[i] = 1.0;

dH1 = H(dK, dX, dY, ciph);

if((dH0 < dHx)&&(dHx < dH1))

{

dH[i] = dH1;

dJ[i] = 1.0;

}

else if((dH1 < dHx)&&(dHx < dH0))

{

dH[i] = dH0;

dJ[i] = 0.0;

}

else

{

dH[i] = dHx;

}

dK[i] = 0.5;

}

else if(dK[i] == 0.0)

{

double dH05, dH1;

dK[i] = 0.5;

dH05 = H(dK, dX, dY, ciph);

dK[i] = 1.0;

dH1 = H(dK, dX, dY, ciph);

if((dH05 < dHx)&&(dHx < dH1))

{

dH[i] = dH1;

dJ[i] = 1.0;

}

else if((dH1 < dHx)&&(dHx < dH05))

{

dH[i] = dH05;

dJ[i] = 0.5;

}

else

{

dH[i] = dHx;

}

dK[i] = 0.0;

}

else if(dK[i] == 1.0)

{

double dH0, dH05;

dK[i] = 0.0;

dH0 = H(dK, dX, dY, ciph);

dK[i] = 0.5;

dH05 = H(dK, dX, dY, ciph);

if((dH0 < dHx)&&(dHx < dH05))

{

dH[i] = dH05;

dJ[i] = 0.5;

}

else if((dH05 < dHx)&&(dHx < dH0))

{

dH[i] = dH0;

dJ[i] = 0.0;

}

else

{

dH[i] = dHx;

}

dK[i] = 1.0;

}

}

dHmax = dH[0];

imax = 0;

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

{

if(dH[i] > dHmax) { dHmax = dH[i]; imax = i; }

}

if(dHmax > dHx)

{

dK[imax] = dJ[imax];

dHx = dHmax;

goto STEP2;

}

return dHmax;

}

typedef struct p

{

double p00, p10, p01, p11;

} probability;

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 dKs[32];

double dX[32];

double dY[32];

double dHx, dHmax;

int j = 0;

probability P[32];

probability Q[32];

unsigned char ucX[5] = "\x22\x6f\x3e\x65";//"0000";

unsigned char ucY[5];

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

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

{

P[i].p00 = 0;

P[i].p01 = 0;

P[i].p10 = 0;

P[i].p11 = 0;

}

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

{

switch (rand()%50)

{

case 0: dKs[i] = 0.0;

break;

case 1: dKs[i] = 1.0;

break;

default: dKs[i] = 0.5;

break;

}

}

/\*\*/

for (int i = 0; i < 1000; i++)

{

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

{

ucK[i] = rand()%0x100;

//ucX[i] = rand()%0x100;

}

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

ucY[i] = ucX[i];

ciph.CryptBlock(ucY,ucK);

//uctod(ucK, dK);

uctod(ucK, dKK);

uctod(ucX, dX);

uctod(ucY, dY);

for (int t = 0; t < 1; t++)

{

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

{

dK[i] = dKs[i];

/\*

switch (rand()%50)

{

case 0: dK[i] = 0.0;

break;

case 1: dK[i] = 1.0;

break;

default: dK[i] = 0.5;

break;

}

/\*\*/

}

dHx = alg1(dK, dX, dY, &ciph);

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

{

if((dK[i] == 0.0)&&(dKK[i] == 0.0)) P[i].p00++;

if((dK[i] == 1.0)&&(dKK[i] == 0.0)) P[i].p10++;

if((dK[i] == 0.0)&&(dKK[i] == 1.0)) P[i].p01++;

if((dK[i] == 1.0)&&(dKK[i] == 1.0)) P[i].p11++;

}

}

}

double temp;

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

{

temp = P[i].p00 + P[i].p01;

Q[i].p00 = P[i].p00/temp;

Q[i].p10 = P[i].p01/temp;

temp = P[i].p10 + P[i].p11;

Q[i].p01 = P[i].p10/temp;

Q[i].p11 = P[i].p11/temp;

temp = P[i].p00 + P[i].p10;

P[i].p00 = P[i].p00/temp;

P[i].p10 = P[i].p10/temp;

temp = P[i].p01 + P[i].p11;

P[i].p01 = P[i].p01/temp;

P[i].p11 = P[i].p11/temp;

}

/\*\*/

cout << fixed ;

cout << setprecision(2);

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

{

cout << i << "\t";

cout << P[i].p00 << "\t";

cout << P[i].p10 << "\t";

cout << P[i].p01 << "\t";

cout << P[i].p11 << endl;

}

cout << endl << endl;

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

{

cout << i << "\t";

cout << Q[i].p00 << "\t";

cout << Q[i].p10 << "\t";

cout << Q[i].p01 << "\t";

cout << Q[i].p11 << endl;

}

cout << endl << endl;

cout.unsetf ( ios\_base::fixed );

\_getch();

return 0;

}