Министерство образования и науки Российской Федерации

Санкт-Петербургский государственный политехнический университет

—

Факультет технической кибернетики

**Кафедра «Информационная безопасность компьютерных систем»**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**Дифференциальный криптоанализ подстановочно-перестановочного шифра**

по дисциплине «Теория итерированных шифров»

Выполнил

студент гр. 5088/1 И.И. Сиваков

<*подпись*>

Проверил

ст. преп. О.В. Шемякина

<*подпись*>

Санкт-Петербург

2011

# Задача

Изучить дифференциальный метод криптоанализа.

# Выполненная работа

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)

Таблица дифференциалов подстановки:

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

------------------------------------------------------------------

0| 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

1| 0 0 0 0 0 0 2 2 0 2 2 0 2 **4** 2 0

2| 0 0 0 2 0 2 0 0 0 **4** 0 2 0 2 0 **4**

3| 0 0 2 **4** 2 0 2 2 0 2 0 0 0 0 2 0

4| 0 0 0 0 0 0 2 2 0 2 2 **4** 2 0 2 0

5| 0 0 2 2 2 2 0 0 0 0 2 2 2 2 0 0

6| 0 0 2 0 2 **4** 2 2 0 2 0 0 0 0 2 0

7| 0 0 2 0 2 0 0 0 0 **4** 2 0 2 0 0 **4**

8| 0 0 0 2 0 2 0 **4** 0 0 0 2 0 2 0 **4**

9| 0 **4** 2 2 0 0 0 0 2 0 2 0 0 2 2 0

10| 0 0 **4** 0 **4** 0 0 0 0 0 **4** 0 **4** 0 0 0

11| 0 **4** 0 2 2 0 0 0 2 0 0 0 2 2 2 0

12| 0 **4** 0 0 2 2 0 0 2 0 0 2 2 0 2 0

13| 0 0 0 2 0 2 **4** 0 **4** 0 0 2 0 2 0 0

14| 0 **4** 2 0 0 2 0 0 2 0 2 2 0 0 2 0

15| 0 0 0 0 0 0 **4** **4** **4** 0 0 0 0 0 0 **4**

Построенная таблица распределения наиболее вероятных дифференциалов по весам:

|  |  |  |
| --- | --- | --- |
| Вес дифференциала | Наибольшая вероятность | Вид дифференциала |
| 2 | 0.000000 |  |
| 3 | 0.250000 | [2, 9]; [9, 1]; [10, 2]; [10, 4]; [12, 1]; |
| 4 | 0.250000 | [1, 13]; [3, 3]; [4, 11]; [6, 5]; [8, 7];  [10, 10]; [10, 12]; [11, 1]; [13, 8]; [14, 1]; |
| 5 | 0.250000 | [2, 15]; [7, 9]; [8, 15]; [13, 6]; [15, 8]; |
| 6 | 0.250000 | [15, 6]; |
| 7 | 0.250000 | [7, 15]; [15, 7]; |
| 8 | 0.250000 | [15, 15]; |

Для вскрытия ключа необходимо построить характеристику, вероятность которой будет максимальна или близка к максимальной.

У подстановки отсутствуют дифференциалы веса 2. Так же никакие 2 активные разряда выхода подстановки не перейдут после перестановки на вход в 1 блок подстановки. Поэтому наиболее вероятная характеристика будет состоять из 4-х дифференциалов с вероятностями 0,25.

S11

S12

S13

S14

S15

S16

S17

S18

Сложение с ключом

Перестановка

S21

S22

S23

S24

S25

S26

S27

S28

Сложение с ключом

Перестановка

S31

S32

S33

S34

S35

S36

S37

S38

Сложение с ключом

Перестановка

Рис.1 построенная характеристика

Вероятность характеристики .

Для вскрытия ключа будем использовать подобранные тексты *x* и *x’*, отличающиеся в двух битах, как показано на рис.1. Для соответствующих шифртекстов *y* и *y’*, будем проводить первый цикл расшифрования. Для этого сгенерируем случайный ключ *k5*. Выполним XOR с шифртекстом, обратную перестановку и подстановку. Ключ, для которого разность полученных значений совпадает с ожидаемой наиболее часто, имеет большую вероятность оказаться истинным.

Была написана программа, выполняющая анализ шифра указанным способом. В результате анализа были вскрыты несколько бит ключа (цифра от 0 до 9 означает вероятность бита быть равным 0 или 1)

01001100 01001100 01001100 01001100 – искомый ключ­­­­

66367436 55546554 37548653 45669643 – полученное значение ключ­­­­а

Как видно, некоторые биты имеют явное тяготение к 1 или 0.

# Выводы

Для вскрытия ключа необходимо около подобранных пар открытых блоков с заданной разностью и соответствующих шифртекстов. Характеристика позволяет вскрыть 8 бит ключа. Необходимо перебрать 8 бит ключа. Тогда вычислительная сложность равна . Для вскрытия всего ключа необходимо, используя другие вероятные характеристики повторить операцию 4 раза.. Вычислительная сложность равна .

Предположим что число активных S блоков на каждом цикле будет равно 1, 1, 2, 2, 2, 2… Тогда вероятность (k-1)-цикловой характеристики будет равна , где k – число циклов шифрования. Для того чтобы анализ имел сложность большую, чем переборная, необходимо . .

# Приложение

//lab3.cpp

#include "stdafx.h"

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;

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

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

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

unsigned char str[] = "Hello world!";

unsigned char key[] = "222222";

cout << ciph.DecryptBlock( ciph.CryptBlock(str, key), key) << endl;

cout << ciph.sub->PrintDiffTable() << endl;

cout << ciph.sub->PrintDweightTable() << endl;

cout << ciph.PrintSubstitution() << endl;

cout << ciph.PrintPermutation() << endl;

ciph.DiffCryptoanalyse(key);

\_getch();

return 0;

}

//Substitution.cpp

#include "cryptoanalyze.h"

#include <math.h>

#include <time.h>

#include <array>

using namespace std;

Substitution::Substitution(int sub\_bits)

{

bits = sub\_bits;

init();

}

void Substitution::GenerateSub(char mode)

{

vector<vector<Pair>> &dweights\_al = dweights;

vector<float> &dprobs\_al = dprobs;

vector<vector<Pair>> &sweights\_al = sweights;

vector<float> &sprobs\_al = sprobs;

int \*temp = new int[size];

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

temp[i] = i;

subs = vector<int>(temp, temp + size);

do

{

do

{

random\_shuffle(subs.begin(), subs.end()) ;

GetDiffTable();

GetSummTable();

GetWeightTable(diff\_table, dweights\_al, dprobs\_al);

GetWeightTable(summ\_table, sweights\_al, sprobs\_al);

}

while(!FastCheck(mode));

CheckSub();

} while (~(~mode|secure)); //implication mode->secure

GetInverce();

delete[] temp;

}

Substitution::Substitution(int sub\_bits, vector<int> sub\_vector)

{

vector<vector<Pair>> &dweights\_al = dweights;

vector<float> &dprobs\_al = dprobs;

vector<vector<Pair>> &sweights\_al = sweights;

vector<float> &sprobs\_al = sprobs;

bits = sub\_bits;

init();

subs = sub\_vector;

GetInverce();

GetDiffTable();

GetSummTable();

GetWeightTable(diff\_table, dweights\_al, dprobs\_al);

GetWeightTable(summ\_table, sweights\_al, sprobs\_al);

CheckSub();

}

Substitution::~Substitution(void)

{

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

{

delete[] diff\_table[i];

delete[] summ\_table[i];

}

delete[] diff\_table;

delete[] summ\_table;

}

void Substitution::GetInverce()

{

int j;

subs\_inv = vector<int>(size);

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

{

for( j = 0; j < size; j++) if (subs[j] == i) break;

subs\_inv[i] = j;

}

}

void Substitution::init()

{

srand (time(NULL));

size = pow( (double)2, bits);

dweights = vector<vector<Pair>>(bits\*2+1);

dprobs = vector<float>(bits\*2+1);

sweights = vector<vector<Pair>>(bits\*2+1);

sprobs = vector<float>(bits\*2+1);

diff\_table = new int\*[size];

summ\_table = new int\*[size];

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

{

diff\_table[i] = new int[size];

summ\_table[i] = new int[size];

}

}

string Substitution::PrintSubstitution()

{

vector<int>::iterator it;

string ret;

char c[100];

ret = "Substitution\n(";

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

{

sprintf(c, "%2d, ", i);

ret += c;

}

ret += "\b\b )\n(";

for (it = subs.begin(); it != subs.end(); it++)

{

sprintf(c, "%2d, ", \*it);

ret += c;

}

ret += "\b\b )\n";

return ret;

}

string Substitution::PrintTable(int \*\*table)

{

string ret = "";

char c[100];

ret += " | ";

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

{

sprintf(c, "%2d ", i);

ret += c;

}

ret += "\n";

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

{

sprintf(c, "----", i);

ret += c;

}

ret += "--\n";

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

{

sprintf(c, "%2d| ", i);

ret += c;

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

{

sprintf(c, "%2d ", table[i][j]);

ret += c;

}

ret += "\n";

}

return ret;

}

string Substitution::PrintWeightTable(vector<vector<Pair>> weights, vector<float> probs)

{

string ret = "";

char c[100];

vector<Pair>::iterator ptr, end;

for (int i = 2; i < bits\*2+1; i++)

{

sprintf(c, " %d %f\t", i, probs[i]);

ret += c;

ptr = weights[i].begin();

end = weights[i].end();

while (ptr != end)

{

sprintf(c, "[%d, %d]; ", ptr->a, ptr->b);

ret += c;

ptr++;

}

ret += "\n";

}

return ret;

}

void Substitution::GetDiffTable()

{

int x1, x2, y1, y2, h1, h2;

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

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

diff\_table[i][j] = 0;

for (x1 = 0; x1 < size; x1++)

{

y1 = subs[x1];

for (x2 = 0; x2 < size; x2++)

{

y2 = subs[x2];

h1 = x1^x2;

h2 = y1^y2;

if ((h1 < size)&&(h2 < size))

diff\_table[h1][h2] ++;

}

}

}

void Substitution::GetSummTable()

{

int x, y, i, j;

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

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

summ\_table[i][j] = 0;

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

{

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

{

for (x = 0; x < size; x++)

{

y = subs[x];

if (( HammingDist(i&x, 0)%2 ) == ( HammingDist(j&y, 0)%2 ))

summ\_table[i][j] ++;

}

summ\_table[i][j] -= pow((double)2, bits-1);

}

}

}

void Substitution::GetWeightTable(int \*\*table, vector<vector<Pair>> &weights, vector<float> &probs)

{

int k;

Pair d;

float temp\_prob;

for( int i = 0; i < 2\*bits+1; i++)

{

weights[i].clear();

probs[i] = 0;

}

for (int i = 1; i < size; i++)

{

for (int j = 1; j < size; j++)

{

k = HammingDist(i, 0) + HammingDist(j, 0);

temp\_prob = (float)abs(table[i][j])/(float)size;

if (temp\_prob != 0)

{

if (temp\_prob > probs[k])

{

probs[k] = temp\_prob;

weights[k].clear();

}

if (temp\_prob == probs[k])

{

d.a = i;

d.b = j;

weights[k].push\_back(d);

}

}

}

}

}

void Substitution::CheckSub()

{

secure = 0xFF;

for (int i = 2; i < 2\*bits+1; i++)

{

if (dprobs[i]\*size > MAX\_PROBABILITY) secure &= ~CHECK\_DIFF;

if (sprobs[i]\*size > MAX\_PROBABILITY) secure &= ~CHECK\_SUMM;

}

#ifdef VERY\_GOOD\_DIFF

if ( dweights[2].size() != 0 ) secure &= ~CHECK\_DWEIGHT;

#else

if ( dprobs[2] > 0.125 ) secure &= ~CHECK\_DWEIGHT;

#endif

#ifdef VERY\_GOOD\_SUMM

if ( sweights[2].size() != 0 ) secure &= ~CHECK\_SWEIGHT;

#else

if ( sprobs[2] > 0.125 ) secure &= ~CHECK\_SWEIGHT;

#endif

secure &= CHECK\_ALL;

}

int Substitution::FastCheck(char mode)

{

#ifdef VERY\_GOOD\_DIFF

if (mode&CHECK\_DWEIGHT) if ( dweights[2].size() != 0 ) return 0;

#else

if (mode&CHECK\_DWEIGHT) if ( dprobs[2] > 0.125 ) return 0;

#endif

#ifdef VERY\_GOOD\_SUMM

if (mode&CHECK\_SWEIGHT) if ( sweights[2].size() != 0 ) return 0;

#else

if (mode&CHECK\_SWEIGHT) if ( sprobs[2] > 0.125 ) return 0;

#endif

if (mode&(CHECK\_DIFF|CHECK\_SUMM))

for (int i = 2; i < 2\*bits+1; i++)

{

if (mode&CHECK\_DIFF) if (dprobs[i]\*size > MAX\_PROBABILITY) return 0;

if (mode&CHECK\_SUMM) if (sprobs[i]\*size > MAX\_PROBABILITY) return 0;

}

return 1;

}

int Substitution::HammingDist(int x, int y)

{

int dist = 0, val = x ^ y;

while(val)

{

++dist;

val &= val - 1;

}

return dist;

}

unsigned char Substitution::SubstituteByte(unsigned char bInput)

{

unsigned char bResult;

bResult = 0x10\*subs[bInput>>4] + subs[bInput & 0x0f];

return bResult;

}

unsigned char Substitution::SubstituteByteInverce(unsigned char bInput)

{

unsigned char bResult;

bResult = 0x10\*subs\_inv[bInput>>4] + subs\_inv[bInput & 0x0f];

return bResult;

}

unsigned char\* Substitution::SubstituteStr(unsigned char\* czInput, int len)

{

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

{

czInput[i] = SubstituteByte(czInput[i]);

}

return czInput;

}

unsigned char\* Substitution::SubstituteStrInverce(unsigned char\* czInput, int len)

{

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

{

czInput[i] = SubstituteByteInverce(czInput[i]);

}

return czInput;

}

//Substitution.h

#pragma once

#include <vector>

#define VERY\_GOOD\_DIFF

//#define VERY\_GOOD\_SUMM

#define MAX\_PROBABILITY 4

#define CHECK\_NO 0x00

#define CHECK\_DIFF 0x01

#define CHECK\_DWEIGHT 0x02

#define CHECK\_SUMM 0x04

#define CHECK\_SWEIGHT 0x08

#define CHECK\_ALL 0x0F

class Substitution

{

public:

Substitution(int sub\_bits = 4);

Substitution(int sub\_bits, std::vector<int> sub\_vector);

~Substitution(void);

private:

typedef struct

{

int a;

int b;

}Pair;

std::string PrintTable(int \*\*table);

std::string PrintWeightTable(std::vector<std::vector<Pair>> weights, std::vector<float> probs);

public:

std::string PrintSubstitution();

std::string PrintDiffTable() { return PrintTable(diff\_table); };

std::string PrintSummTable() { return PrintTable(summ\_table); };

std::string PrintDweightTable(){ return PrintWeightTable(dweights, dprobs); };

std::string PrintSweightTable(){ return PrintWeightTable(sweights, sprobs); };

void GenerateSub(char mode = CHECK\_ALL);

unsigned char SubstituteByte(unsigned char bInput);

unsigned char SubstituteByteInverce(unsigned char bInput);

unsigned char\* Substitution::SubstituteStr(unsigned char\* x, int len);

unsigned char\* Substitution::SubstituteStrInverce(unsigned char\* czInput, int len);

private:

void init();

void GetInverce();

void GetDiffTable();

void GetSummTable();

void GetWeightTable(int \*\*table, std::vector<std::vector<Pair>> &weights, std::vector<float> &probs);

void CheckSub();

int FastCheck(char mode);

int HammingDist(int x, int y);

std::vector<int> subs, subs\_inv;

public:

int size;

int bits;

int \*\*diff\_table;

int \*\*summ\_table;

std::vector<std::vector<Pair>> dweights;

std::vector<float> dprobs;

std::vector<std::vector<Pair>> sweights;

std::vector<float> sprobs;

char secure;

};

#include "cryptoanalyze.h"

#include <algorithm>

#include <iostream>

#include <math.h>

#include <time.h>

using namespace std;

Cipher::Cipher(vector<int> svector, vector<int> pvector)

{

sub = new Substitution(4, svector);

per = new Permutation(32, pvector);

}

Cipher::~Cipher(void)

{

}

unsigned char \*Cipher::CryptBlock(unsigned char\* czInBlock, unsigned char\* czKeyBlock)

{

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

{

for (int i = 0; i < 4; i++) czInBlock[i]^=czKeyBlock[i];

czInBlock = sub->SubstituteStr(czInBlock, 4);

czInBlock = per->PermutateBlock(czInBlock);

}

for (int i = 0; i < 4; i++) czInBlock[i]^=czKeyBlock[i];

return czInBlock;

}

unsigned char \*Cipher::CryptCycleBlock(unsigned char\* czInBlock, unsigned char\* czKeyBlock)

{

for (int i = 0; i < 4; i++) czInBlock[i]^=czKeyBlock[i];

czInBlock = sub->SubstituteStr(czInBlock, 4);

czInBlock = per->PermutateBlock(czInBlock);

return czInBlock;

}

unsigned char \*Cipher::DecryptCycleBlock(unsigned char\* czInBlock, unsigned char\* czKeyBlock)

{

for (int i = 0; i < 4; i++) czInBlock[i]^=czKeyBlock[i];

czInBlock = per->PermutateBlockInverce(czInBlock);

czInBlock = sub->SubstituteStrInverce(czInBlock, 4);

return czInBlock;

}

unsigned char \*Cipher::DecryptBlock(unsigned char\* czInBlock, unsigned char\* czKeyBlock)

{

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

{

for (int i = 0; i < 4; i++) czInBlock[i]^=czKeyBlock[i];

czInBlock = per->PermutateBlockInverce(czInBlock);

czInBlock = sub->SubstituteStrInverce(czInBlock, 4);

}

for (int i = 0; i < 4; i++) czInBlock[i]^=czKeyBlock[i];

return czInBlock;

}

typedef struct

{

int a;

int b;

float prob;

}DIFF;

bool compare (DIFF i,DIFF j) { return (i.prob>j.prob); }

unsigned char \*Cipher::DiffCryptoanalyse(unsigned char\* szKeyBlock)

{

unsigned char szKeyBlock2[4];

for (int i = 0; i < 4; i++) szKeyBlock2[i] = szKeyBlock[i];

for (int i = 0; i < 4; i++) {for (int j = 0; j < 8; j++) {printf("%d", szKeyBlock2[i]%2); szKeyBlock2[i] /=2;} printf(" ");}

printf("\n---------------------------------\n");

srand(time(NULL));

double dBits[32] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

double num = 0;

unsigned char x1[4];

unsigned char y1[4];

unsigned char x2[4];

unsigned char y2[4];

unsigned char z1[4];

unsigned char z2[4];

unsigned char a1[4];

unsigned char b1[4];

unsigned char tempKey[4];

a1[0] = 0;

a1[1] = 0;

a1[2] = 0;

a1[3] = 10;

b1[0] = 16;

b1[1] = 130;

b1[2] = 16;

b1[3] = 192;

for(;;)

{

for (int i = 0; i < 4; i++) x1[i] = (unsigned char) rand();

for (int i = 0; i < 4; i++) x2[i] = (x1[i]^a1[i]);

for (int i = 0; i < 4; i++) {y1[i] = x1[i]; y2[i] = x2[i];}

CryptBlock(y1, szKeyBlock);

CryptBlock(y2, szKeyBlock);

for (int g = 0; g< 10000; g++)

{

for (int i = 0; i < 4; i++) tempKey[i] = (unsigned char) rand();

for (int i = 0; i < 4; i++) {z1[i] = y1[i]^tempKey[i]; z2[i] = y2[i]^tempKey[i];}

per->PermutateBlockInverce(z1);

sub->SubstituteStrInverce(z1, 4);

per->PermutateBlockInverce(z2);

sub->SubstituteStrInverce(z2, 4);

if(!(b1[0]^z1[0]^z2[0]))

if(!(b1[1]^z1[1]^z2[1]))

if(!(b1[2]^z1[2]^z2[2]))

if(!(b1[3]^z1[3]^z2[3]))

{

num++; for (int i = 0; i < 4; i++) {for (int j = 0; j < 8; j++) {dBits[i\*8+j] += tempKey[i]%2; tempKey[i] /=2; printf("%0.0f", 10\*dBits[i\*8+j]/num);} printf(" ");}

printf("\n");

break;

}

}

}

return NULL;

}