

Министерство образования и науки РФ  
Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования

Тульский государственный университет

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

## **МЕТОДЫ КРИПТОГРАФИИ ПОДСТАНОВКИ**

Лабораторная работа № 1  
по курсу «Методы и средства защиты компьютерной информации»

Выполнил:	студент группы 220601	_____	Белым А.А.
		(подпись)	
Проверил:	д. т. н., проф. каф. ВТ	_____	Данилкин Ф.А.
		(подпись)	

Тула 2013

## Цель работы

Знакомство с методами подстановок в криптографии. Получение навыков шифрования и дешифрования сообщений, а также разработки соответствующего программного обеспечения.

## Задание

Разработать программу для шифрации и дешифрации текста из заданного файла с использованием XOR-шифрования, а также взлома зашифрованного текста.

## Текст программы

Далее представлен текст программы на языке C++, реализующей шифрование, дешифрование и взлом файла с текстом.

```
#include <iostream>
#include <unordered_map>
#include <functional>
#include <fstream>
#include <algorithm>
#include <string>

using namespace std;
struct char_array{
    char *data;
    size_t len;
};

void xor_crypt(char_array plain,char_array key_, char_array& res){
    if(plain.len>res.len)
        return;
    char *end=plain.data+plain.len,
        *key_start=key_.data,
        *key_end=key_.data+key_.len;
    char *ptr=res.data;
    while(plain.data!=end){
        if(key_.data==key_end)
            key_.data=key_start;
        *ptr=*plain.data^*key_.data;
        ptr++;plain.data++;key_.data++;
    }
}

char_array xor_crypt(char *text,char *key_){
    char_array text_arr={text,strlen(text)},
        key_arr={key_,strlen(key_)},
        res={new char[text_arr.len],text_arr.len};
    xor_crypt(text_arr,key_arr,res);
    return res;
}

char_array xor_crypt(char_array encrypted,char *key_){
    char_array res={new char[encrypted.len],encrypted.len},
        key_arr={key_,strlen(key_)};
    xor_crypt(encrypted,key_arr,res);
    return res;
}
```

```

ostream& operator <<(ostream &os,char_array a){
    char *end=a.data+a.len;
    for(;a.data!=end;++a.data)
        os<<* (a.data) ;
    return os;
}

typedef unordered_map<char,double> letter_table;
typedef unordered_map<string,double> bigram_table;

letter_table get_letter_table(char_array text,size_t step=1,size_t start=0){
    letter_table table;size_t len=0;
    for(size_t i=start;i<text.len;i+=step){
        ++table[text.data[i]];
        ++len;
    }
    for(auto i=table.begin();i!=table.end();i++){
        i->second/=len;
    }
    return table;
}

bigram_table get_bigram_table(char_array text,size_t step=1,size_t start=0){
    bigram_table table;size_t len=0;
    string t;
    for(size_t i=start;i<(text.len-1);i+=step){
        t=text.data[i];
        t+=text.data[i+1];
        ++table[t];
        ++len;
    }
    for(auto i=table.begin();i!=table.end();i++){
        i->second/=len;
    }
    return table;
}

struct letter_freq{
    char c; double freq;
};
struct bigram_freq{
    char c[2]; double freq;
};
template <typename T>
bool freq_comparer(const T&a, const T& b){
    return a.freq>b.freq;
}
#define delta(a,b) (a>b?a-b:b-a)

class Cracker{
private:
    letter_table t; bigram_table t2;
    char_array enc;
    size_t key_len; char *key_;
    static void fill_ltr_vector(const letter_table& t, vector<letter_freq> &v){
        letter_freq lfq;
        for(auto i= t.begin();i!=t.end();i++){
            lfq.c=i->first;
            lfq.freq=i->second;
            v.push_back(lfq);
        }
        sort(v.begin(),v.end(),freq_comparer<letter_freq>);
    }
    static void fill_bgrm_vector(const bigram_table& t,vector<bigram_freq>& v){
        bigram_freq bfq;
        for(auto i= t.begin();i!=t.end();i++){

```

```

        bfq.c[0]=i->first[0];
        bfq.c[1]=i->first[1];
        bfq.freq=i->second;
        v.push_back(bfq);
    }
    sort(v.begin(),v.end(),freq_comparer<bigram_freq>);
}

static void fill_bgrm_vector(const bigram_table& t,char c,vector<bigram_freq>&
v){
    bigram_freq bfq;
    for(auto i= t.begin();i!=t.end();i++){
        if(i->first[0]==c){
            bfq.c[0]=i->first[0];
            bfq.c[1]=i->first[1];
            bfq.freq=i->second;
            v.push_back(bfq);
        }
    }
    sort(v.begin(),v.end(),freq_comparer<bigram_freq>);
}
Cracker(const Cracker&);
public:
double ltr_eps,ltr_min,bgrm_eps,bgrm_min;
int default_key_len;

typedef bool (*check_key_callb) (Cracker*);
check_key_callb check_key;

Cracker(){
    bgrm_eps=0.05;
    bgrm_min=0.01;
    ltr_eps=0.05;
    ltr_min=0.01;
    default_key_len=3;
}

inline char* key(){return key_;}
inline size_t key_len(){return key_len_;}
inline char_array encrypted_text(){return enc;}

bool crack(char_array enc_,const letter_table& t_,const bigram_table& t2_){
    t=t_; t2=t2_; enc=enc_;
    key_len_=default_key_len;
    vector<letter_freq> v1,v2;
    fill_ltr_vector(t,v2);

    bool cont=true,try_bgr=true;
    while(cont&&(enc.len>key_len_)){
        letter_table tmp=get_letter_table(enc,key_len_);
        fill_ltr_vector(tmp,v1);
        key_=new char[key_len_+1];
        for(auto i=v1.begin();i!=v1.end()&&i->freq>ltr_min;++i){
            auto j=v2.begin();
            while(j!=v2.end()&&delta(i->freq,j->freq)>ltr_eps)
                ++j;

            while(j!=v2.end()&&delta(i->freq,j->freq)<=ltr_eps&&j-
>freq>ltr_min){
                key_[0]=i->c^j->c;
                try_bgr=true;
                for(auto k=tmp.begin();k!=tmp.end();k++){
                    if(!t[(k->first)^key_[0]]){
                        try_bgr=false;
                        break;

```

```

        }

        }
        if(try_bgr){
            cont=find_bigram(1);
            if(!cont)
                return true;
        }
        ++j;
    }
}
delete[] key_;
key_len_++;
}
return false;
}
bool find_bigram(int pos){
    bigram_table tmp=get_bigram_table(enc,key_len_,pos-1);
    vector<bigram_freq> v1,v2;
    fill_bgrm_vector(tmp,v1);

    char c1,c2; bool cont=true,try_bgr=true;
    for(auto i=v1.begin();i!=v1.end()&&i->freq>bgrm_min;++i){
        c1=i->c[0]^key_[pos-1];
        fill_bgrm_vector(t2,c1,v2);

        auto j=v2.begin();
        while(j!=v2.end()&&delta(i->freq,j->freq)>bgrm_eps)
            ++j;

        while(j!=v2.end()&&delta(i->freq,j->freq)<=bgrm_eps&&
            j->freq>bgrm_min){
            c2=j->c[1];
            key_[pos]=c2^i->c[1];
            try_bgr=true; string s;
            for(auto k=tmp.begin();k!=tmp.end();k++){
                s=(k->first[0])^key_[pos-1];
                s+=(k->first[1])^key_[pos];
                if(!t[s[1]]||!t2[s]){
                    try_bgr=false;
                    break;
                }
            }
            if(try_bgr){
                if(pos==key_len_-1){
                    key_[key_len_]='\0';
                    cont=check_key(this);
                }
                else
                    cont=find_bigram(pos+1);
                if(!cont)
                    return false;
            }
            ++j;
        }
    }
    return cont;
}
};

bool check_key(Cracker* cracker){
    cout<<"Found a possible key: ";
    cout<<cracker->key()<<endl;
    cout<<"Decrypted text is: "<<endl<<endl;
    char_array ch=xor_crypt(cracker->encrypted_text(),cracker->key());
    cout<<ch<<endl<<endl;
}

```

```

        cout<<"Stop search?[Y/N] ";
        string answ;
        cin>>answ;
        delete[] ch.data;
        return answ=="N";
    }

bigram_table read_bigram(string fname){
    bigram_table t;
    ifstream ifs;
    ifs.open(fname,ios::binary);
    string s;
    char ch1,ch2;double d;
    while(ifs.good()){
        ifs.read(&ch1,sizeof(ch1));
        ifs.read(&ch2,sizeof(ch2));
        ifs.read((char*)&d,sizeof(d));
        s=ch1;s+=ch2;
        t[s]=d;
    }
    ifs.close();
    return t;
}

void write_bigram(const bigram_table& t,string fname){
    ofstream ofs;
    ofs.open(fname,ios::binary);
    for(auto i=t.begin();i!=t.end();++i){
        ofs.write(i->first.c_str(),2);
        ofs.write((char*)&(i->second),sizeof(i->second));
    }
    ofs.close();
}

letter_table read_letter(string fname){
    letter_table lt;
    ifstream ifs;
    ifs.open(fname,ios::binary);
    char ch;double d;
    while(ifs.good()){
        ifs.read(&ch,sizeof(ch));
        ifs.read((char*)&d,sizeof(d));
        lt[ch]=d;
    }
    ifs.close();
    return lt;
}

void write_letter(const letter_table& t,string fname){
    ofstream ofs;
    ofs.open(fname,ios::binary);
    for(auto i=t.begin();i!=t.end();++i){
        ofs.write(&(i->first),sizeof(i->first));
        ofs.write((char*)&(i->second),sizeof(i->second));
    }
    ofs.close();
}

void write_char_array(char_array arr,string fname){
    ofstream ofs;
    ofs.open(fname);
    ofs.write(arr.data,arr.len);
    ofs.close();
}

char_array read_char_array(string fname){
    ifstream ifs;

```

```

    ifs.open(fname);
    char_array arr;
    ifs.seekg(0, std::ios::end);
    arr.len=ifs.tellg();
    arr.data=new char[arr.len+1];
    arr.data[arr.len]=0;
    ifs.seekg(0, std::ios::beg);
    ifs.read(arr.data, arr.len);
    ifs.close();
    return arr;
}

void usage(){
    cout<<
        "cryptol - Encryption/decryption and cracking of XOR-
encryption."<<endl<<
        "Usage:"<<endl<<
        "cryptol crypt KEY [INFILE] [OUTFILE]"<<endl<<
        "    Encrypts or decrypts with given KEY contents of INFILE"<<endl<<
        "    and writes result to OUTFILE."<<endl<<
        "    By default: INFILE=\"input.txt\"; OUTFILE=\"output.txt\""<<endl<<
        endl<<
        "cryptol analyze [INFILE] [LETTERSFILE] [BIGRAMSFILE]"<<endl<<
        "    Do a frequency analysis of letters and bigrams in contents of
INFILE,"<<endl<<
        "    and writes resulting tables to LETTERSFILE and BIGRAMSFILE
respectively."<<endl<<
        "    By default: INFILE=\"input.txt\"; LETTERSFILE=\"letters.txt\";
BIGRAMSFILE=\"bigrams.txt\";"<<endl<<
        endl<<
        "cryptol crack [INFILE] [OUTFILE] [KEYFILE] [LETTERSFILE]
[BIGRAMSFILE]"<<endl<<
        "    Cracks and finds encrypted contents of INFILE"<<endl<<
        "    and writes plain result to OUTFILE."<<endl<<
        "    If KEYFILE is not empty, writes founded key to KEYFILE."<<endl<<
        "    Cracking uses tables of a frequency analysis of letters and
bigrams"<<endl<<
        "    from files LETTERSFILE and BIGRAMSFILE."<<endl<<
        "    By default: INFILE=\"input.txt\"; OUTFILE=\"output.txt\";
KEYFILE=\"\"<<endl<<
        "    LETTERSFILE=\"letters.txt\";
BIGRAMSFILE=\"bigrams.txt\";"<<endl<<
    }

int main(int argc,char **argv)
{
    string infile="input.txt",
        outfile="output.txt",
        letters="letters.txt",
        bigrams="bigrams.txt",
        keyfile="";

    int i=0;
    char *key;
    --argc;i++;
    if(argc<1){
        usage();
        return -1;
    }
    --argc;i++;
    string s;
    s=argv[1];
    if(s=="crypt"){
        if (argc>3||argc<1){
            usage();
            return -1;
        }
        key=argv[i];

```

```

        --argc;i++;
        if(argc>0){
            infile=argv[i];
            --argc;i++;
        }
        if(argc>0){
            outfile=argv[i];
            --argc;i++;
        }
    } else if(s=="analyze"){
        if (argc>3){
            usage();
            return -1;
        }
        if(argc>0){
            infile=argv[i];
            --argc;i++;
        }
        if(argc>0){
            letters=argv[i];
            --argc;i++;
        }
        if(argc>0){
            bigrams=argv[i];
            --argc;i++;
        }
    } else if(s=="crack"){
        if (argc>5){
            usage();
            return -1;
        }
        if(argc>0){
            infile=argv[i];
            --argc;i++;
        }
        if(argc>0){
            outfile=argv[i];
            --argc;i++;
        }
        if(argc>0){
            keyfile=argv[i];
            --argc;i++;
        }
        if(argc>0){
            letters=argv[i];
            --argc;i++;
        }
        if(argc>0){
            bigrams=argv[i];
            --argc;i++;
        }
    }
}

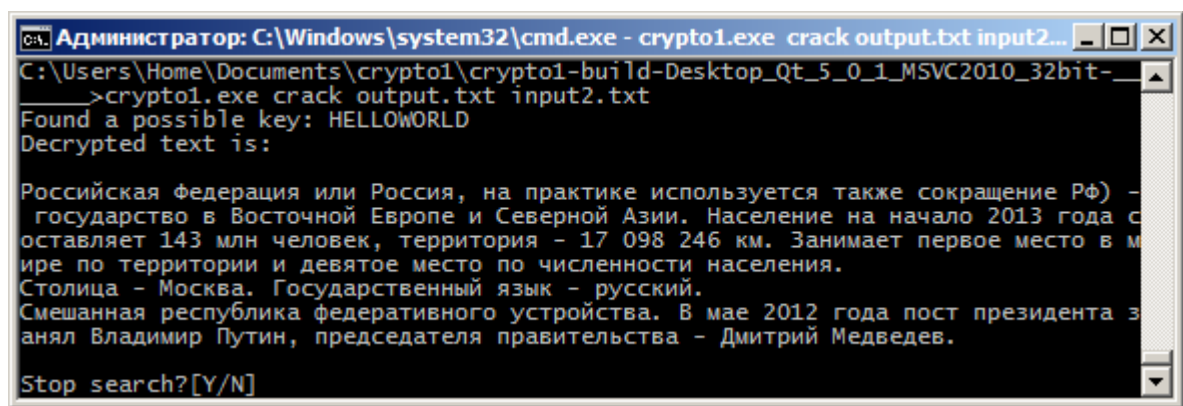
if(s=="crypt"){
    char_array src=read_char_array(infile),
               res=xor_crypt(src,key);
    write_char_array(res,outfile);
    delete[] src.data; delete[] res.data;
} else if(s=="analyze"){
    char_array src=read_char_array(infile);
    letter_table lt=get_letter_table(src);
    bigram_table bt=get_bigram_table(src);

    write_letter(lt,letters);
    write_bigram(bt,bigrams);
    delete[] src.data;
}

```







```
Администратор: C:\Windows\system32\cmd.exe - crypto1.exe crack output.txt input2...
C:\Users\Home\Documents\crypto1\crypto1-build-Desktop_Qt_5_0_1_MSVC2010_32bit->
>crypto1.exe crack output.txt input2.txt
Found a possible key: HELLOWORLD
Decrypted text is:
Российская Федерация или Россия, на практике используется также сокращение РФ) –
государство в Восточной Европе и Северной Азии. Население на начало 2013 года с
оставляет 143 млн человек, территория – 17 098 246 км. Занимает первое место в м
ире по территории и девятое место по численности населения.
Столица – Москва. Государственный язык – русский.
Смешанная республика федеративного устройства. В мае 2012 года пост президента з
анял Владимир Путин, председателя правительства – Дмитрий Медведев.
Stop search?[Y/N]
```

Рисунок 1— Пример работы программы при взломе

После выполнения программы, в файл input2.txt записывается исходный текст.

### Вывод

В данной работе я познакомился с различными подстановочными шифрами. Подстановочные шифры широко использовались в докомпьютерную эпоху, однако большая часть этих шифров уязвима к взлому методом частотного анализа. Была написана программа, которая может шифровать текст, и также дешифровывать или взламывать зашифрованный текст.