STATE COMMITTEE OF COMMUNICATION, INFORMATIZATION AND TELECOMMUNICATION TECHNOLOGIES OF THE REPUBLIC OF UZBEKISTAN
TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES

Coursework

Systemic theoretical approach to the design stream ciphers

Performed By: Zarbiev V .233-12.
Adopted By : Gulomov Sh.R.

-Tashkent 2014-

CONTENT

1 Introduction

Stream codes on the basis of shift registers were actively used in the years of war, even long before emergence of electronics. They were simple in design and realization.

In 1965 Ernst Selmer, the main cryptographer of the Norwegian government, developed the theory of sequence of shift registers. Later Solomon Golomb, the mathematician of the U.S. National Security Agency, wrote the book under the name "Shift Register Sequences" ("Sequences of shift registers") in which stated the main achievements in this area, and also Selmer's achievements.

The great popularity to stream codes was brought by Claude Shannon's work published in 1949 in which Shannon proved absolute firmness of the code of Vernam (also known as a disposable notebook). In Vernam's code the key has length equal to length of the most transferred message. The key is used as scale and if each bit of a key gets out incidentally, it is impossible to open the code (since all possible clear texts will be ravnoveroyatna). A large number of algorithms of stream enciphering, such as is so far created: A3, A5, A8, MUGI, PIKE, RC4, SEAL, ORION.

2 The stream code

The stream code — is the symmetric code in which each symbol of a clear text will be transformed to a text in code symbol in dependence not only on the used key, but also on its arrangement in a clear text stream. The line code realizes other approach to symmetric enciphering, than block codes

3 The main differences of line codes from the block

The majority of the existing codes with a confidential key unambiguously can be carried either to line, or to block codes. But the theoretical border between them is quite indistinct. For example, algorithms of block enciphering in the mode of line enciphering (an example are used: for algorithm of Desrezhimy CFB and OFB).

We will consider the main distinctions between line and block codes not only in aspects of their safety and convenience, but also from the point of view of their studying in the world:

• the most important advantage of line codes before block is the high speed of enciphering, commensurable with a speed of receipt of entrance information; therefore, enciphering practically in real time regardless of volume and word length of a flow of the transformed data is provided.

• in synchronous line codes (unlike block) there is no effect of reproduction of mistakes, that is number of the distorted elements in the deciphered sequence is equal to number of the distorted elements of the ciphered sequence which came from a communication channel.

• the structure of a line key can have weak spots which give the chance to the cryptanalyst to receive additional information on a key (for example, at the small period of a key the cryptanalyst can use the found parts of a line key for decoding of the subsequent closed text).

• PSh unlike BSh can be often attacked by means of linear algebra (as releases of separate registers of shift with the return linear communication can have correlation with scale). Also the linear and differential analysis is quite successfully applied to breaking of line codes.

Now about situation in the world:

• in the majority of works on the analysis and breaking of block codes the algorithms of enciphering based on the DES standard are considered; for line codes there is no the allocated direction of studying; methods of breaking of PSh are very various.

• for line codes the set of the requirements which are criteria of reliability (the big periods of output sequences, postulates of Golomba, nonlinearity) is established; for BSh such accurate criteria aren't present.

• the European cryptographic centers, block – American generally are engaged in research and development of line codes.

• research of line codes happens more dynamically, than block; recently it wasn't made any noticeable discoveries in the sphere of DES algorithms while in the field of line codes there was a set of progress and failures (some schemes seeming resistant at further research didn't equal hopes of inventors).

4 Design of line codes

According to Rainer Rueppel it is possible to allocate four main approaches to design of line codes:

• System and theoretical approach is based on creation for the cryptanalyst of difficult, earlier unexplored problem.

• Slozhnostno-teoretichesky approach is based on a difficult, but known problem (for example, factorization of numbers or discrete logarithming).

• Information and technical approach is based on attempt to conceal a clear text from the cryptanalyst – regardless of that is what is the time spent for decoding, the cryptanalyst won't find the unambiguous solution.

• Randomized approach is based on creation of a volume task; the cryptographer thereby tries to make the solution of a problem of deciphering physically impossible. For example, the cryptographer can cipher some article, and instructions on will be a key what parts of article were used when enciphering. The cryptanalyst should touch all casual combinations of parts of article before to him carries, and he will define a key.

Theoretical criteria of Rainer Rueppel for design of line systems:

• long periods of output sequences;

• big linear complexity;

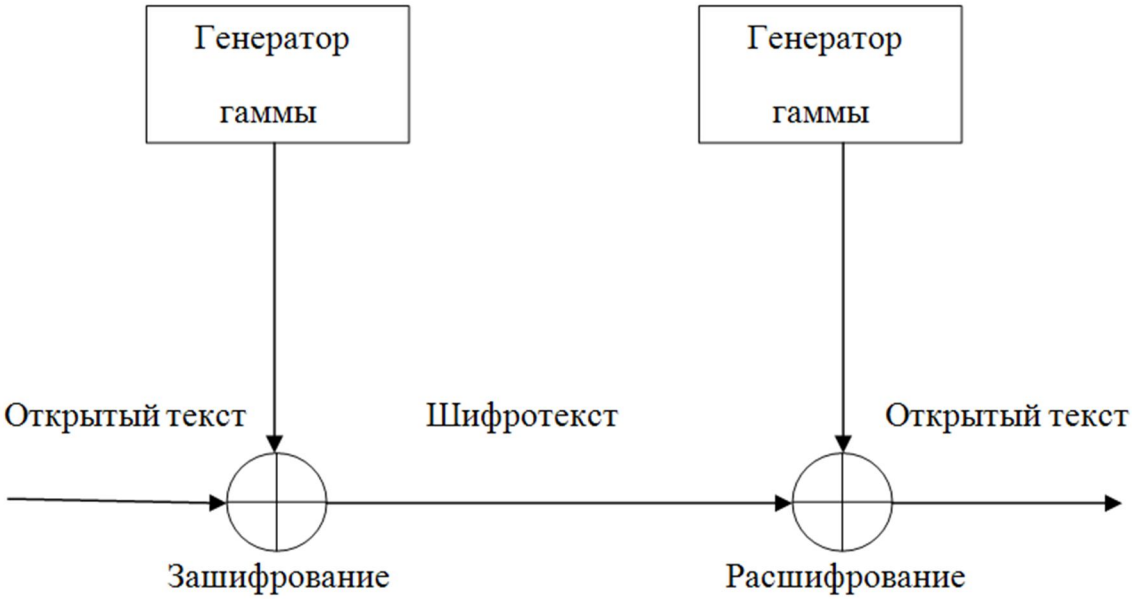
• diffusion – dispersion of redundancy in substructures, "spreading" of statistics on all text;

• each bit of a stream of keys has to be difficult transformation of the majority of bits of a key;

• criterion of nonlinearity for logical functions.

Still it isn't proved that these criteria are necessary or sufficient for safety of line system of enciphering. It is also worth noticing that if the cryptanalyst possesses unlimited time and computing power, the only realized stream code protected from such opponent is the disposable notebook.

The simplest implementation of the line code is represented in drawing.



The generator of scale gives out a key stream (scale):. We will designate a stream of bits of a clear text. Then the stream of bits of a shifrotekst turns out by means of operation XOR application: where.

Deciphering is made by the operation XOR between the same scale and the ciphered text:.

It is obvious that if the sequence of bits of scale has no period and gets out incidentally, it is impossible to hack the code. But this mode of enciphering has also negative features. So keys, comparable on length with the transferred messages, it is difficult to use in practice. Therefore usually apply a key of smaller length (for example, 128 bits). By means of it the pseudorandom gammiruyushchy sequence is generated (it has to satisfy to postulates of Golomba). Naturally, pseudo-accident of scale can be used at attack to the stream code.

5 Generator of pseudo random numbers of Shamir

Edie Shamir used algorithm of RSA as the generator of pseudorandom numbers. Though Shamir showed that a prediction of an exit of the generator of pseudorandom numbers equivalently to breaking of RSA.

Cryptographic systems with an open key use so-called unilateral functions which possess the following property:

If it is known x, то $f(x)$ to calculate rather simply

If it is known $y = f(x)$, that for calculation isn't present a simple (effective) way.

The unilaterality is understood as not a theoretical one-orientation, but practical impossibility to calculate the return value, using modern computing means, for a foreseeable interval of time.

Complexity of a problem of factorization of work of two large prime numbers is the basis for cryptographic system with an open key of RSA. For enciphering transaction of exponentiation of the module of a large number is used. For decoding for reasonable time (the return operation) it is necessary to be able to calculate Euler's function from this large number for what it is necessary to know decomposition of number on simple multipliers.

In cryptographic system with an open key each participant has as an open key (English public key), and the closed key (English private key). In cryptographic RSA system each key consists of couple of integers. Each participant creates the opened and closed key independently. The closed key keeps each of them a secret, and open keys can be told anyone or even to publish them. The opened and closed keys of each participant of an exchange of messages in RSA cryptosystem form "the coordinated couple" in the sense that they are the mutually return, that is:

admissible couples of opened and closed keys (p, s) $\exists \quad m = D_s(E_p(m)) = E_p(D_s(m))$. enciphering functions $E_p(x)$ and decipherings $D_s(x)$ such that messages $\forall m \in M$, где M — set of admissible messages.

Algorithm of creation of open and confidential keys [to govern RSA keys are generated as follows:

Two various casual prime numbers and a given size get out (for example, 1024 bits everyone).

Work is calculated them , which is called as the module.

Value of function of Euler is calculated

8 System and theoretical approach to design of stream codes

In practice, design of the stream code in many respects similar design of the block code. In this case more mathematical theory is used, but eventually the cryptographer offers some scheme and then tries to make its analysis.

According to Rainer Rueppel there are four various approaches to design of stream codes:

— System and theoretical approach. Using a number of fundamental criteria and laws of design, tries to make sure that each scheme creates a complex and unknown problem for the cryptanalyst.

— Information and theoretical approach. Tries to keep a clear text unknown to the cryptanalyst. Irrespective of the fact how many actions will be executed by the cryptanalyst, he will never receive odnoznachnogoresheniye.

— Slozhnostno-teoretichesky approach. Tries to use as the basis for cryptosystem some known and complex problem, such as decomposition on multipliers or a capture of discrete logarithms, or to make cryptosystem to equivalent this problem.

— Randomized approach. Tries to create extremely big problem, forcing the cryptanalyst to check a set of senseless data during cryptoanalysis attempts.

These approaches differ in assumptions of opportunities and abilities of the cryptanalyst, definition of success of cryptoanalysis and understanding of safety. The majority of researches in this area - theoretical, but among useless stream codes are also the quite decent.

System and theoretical approach was used in all earlier provided stream codes, its applications are result the majority of the stream codes used in the real world. The cryptographer develops the generators of a stream of keys possessing the checked

characteristics of safety - the period, distribution of bits, linear complexity, etc. - but not the codes based on the mathematical theory. The cryptographer also studies various methods of cryptoanalysis of these generators and checks, whether are steady a gene e-ratory in relation to these ways of opening.

Over time this approach led to emergence of a set of criteria of design of stream codes. They were considered by Ryuppel it in detail gives theoretical bases of these criteria.

— The long period without repetitions.

— Criterion of linear complexity - big linear complexity, a linear profile of complexity, local linear complexity, etc.

— Statistical criteria, for example, ideal A:mernyeraspredeleniya.

— Confusion - each bit of a stream of keys has to be difficult transformation of all or the majority of bits of a key.

— Diffusion - redundancy in substructures has to dissipate, leading to more "smeared" statistics.

— Criteria of nonlinearity for logical functions, such as lack of correlation of t-go of an order, distance before linear functions, avalanche criterion, etc.

This list of criteria of design isn't unique for the stream codes developed by means of system and theoretical approach, it is fair for all stream codes. It is fair and for all block codes. Feature of system and theoretical approach is that stream codes are directly developed to satisfy to these criteria.

The main problem of such cryptosystems is the impossibility to prove their safety, was never proved that these criteria of design are necessary or sufficient for safety. The generator of a stream of keys can satisfy to all rules of development, but nevertheless be unsafe. Another can be safe. This process still there is something magic.

On the other hand opening of any of these generators of a stream of keys represents an excellent problem for the cryptanalyst. If enough various generators are developed, it can appear that the cryptanalyst won't begin to spend time, hacking each of them. Perhaps he will be interested more by opportunity to become

famous, having achieved success, factorizing large numbers or calculating discrete logarithms.

7 Euler's function in RSA

Setting up an RSA system involves choosing large prime numbers p and q, computing $n = pq$ and $k = \varphi(n)$, and finding two numbers e and d such that $ed \equiv 1 \pmod{k}$. The numbers n and e (the "encryption key") are released to the public, and d (the "decryption key") is kept private.

A message, represented by an integer m, where $0 < m < n$, is encrypted by computing $S = m^e \pmod{n}$.

It is decrypted by computing $t = S^d \pmod{n}$. Euler's Theorem can be used to show that if $0 < t < n$, then $t = m$.

The security of an RSA system would be compromised if the number n could be factored or if $\varphi(n)$ could be computed without factoring n.

8 Code of RSA

RSA involves a *public key* and a *private key*. The public key can be known by everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.
   - For security purposes, the integers p and q should be chosen at random, and should be of similar bit-length. Prime integers can be efficiently found using a primality test.
2. Compute $n = pq$.
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.

3. Compute $\varphi(n) = \varphi(p)\varphi(q) = (p - 1)(q - 1) = n - (p + q - 1)$, where φ is Euler's totient function.
4. Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$; i.e., e and $\varphi(n)$ are coprime.
   - e is released as the public key exponent.
   - e having a short bit-length and small Hamming weight results in more efficient encryption – most commonly $2^{16} + 1 = 65{,}537$. However, much smaller values of e (such as 3) have been shown to be less secure in some settings.[5]
5. Determine d as $d \equiv e^{-1} \pmod{\varphi(n)}$; i.e., d is the multiplicative inverse of e (modulo $\varphi(n)$).

   - This is more clearly stated as: solve for d given $d \cdot e \equiv 1 \pmod{\varphi(n)}$
   - This is often computed using the extended Euclidean algorithm. Using the pseudocode in the *Modular integers* section, inputs a and n correspond to e and $\varphi(n)$, respectively.
   - d is kept as the private key exponent.

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the modulus n and the private (or decryption) exponent d, which must be kept secret. p, q, and $\varphi(n)$ must also be kept secret because they can be used to calculate d.

- An alternative, used by PKCS#1, is to choose d matching $de \equiv 1 \pmod{\lambda}$ with $\lambda = \operatorname{lcm}(p - 1, q - 1)$, where lcm is the least common multiple. Using λ instead of $\varphi(n)$ allows more choices for d. λ can also be defined using the Carmichael function, $\lambda(n)$.
- The ANSI X9.31 standard prescribes, IEEE 1363 describes, and PKCS#1 allows, that p and q match additional requirements: being strong primes, and being different enough that Fermat factorization fails.

Encryption[edit]

Alice transmits her public key (*n*, *e*) to Bob and keeps the private key *d* secret. Bob then wishes to send message *M* to Alice.

He first turns *M* into an integer *m*, such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext *c*corresponding to

$$c \equiv m^e \pmod{n}$$

This can be done efficiently, even for 500-bit numbers, using Modular exponentiation. Bob then transmits *c* to Alice.

Note that at least nine values of *m* will yield a ciphertext *c* equal to *m*,[note 1] but this is very unlikely to occur in practice.

Decryption[edit]

Alice can recover *m* from *c* by using her private key exponent *d* via computing

$$m \equiv c^d \pmod{n}$$

Given *m*, she can recover the original message *M* by reversing the padding scheme.

(In practice, there are more efficient methods of calculating c^d using the precomputed values below.)

9 Vizhener's code

At each stage of enciphering various alphabets chosen depending on a keyword symbol are used. For example, suppose, that the source text has an appearance:

ATTACKATDAWN

The person sending the message writes down a keyword ("LEMON") cyclically until its length doesn't correspond to length of a source text:

LEMONLEMONLE

The first symbol of a source text of A is ciphered by sequence of L which is the first symbol of a key. The first symbol of the L text in code is on crossing of a line L and column A in Vizhener's table. In the same way for the second symbol of a source text the second symbol of a key is used; that is the second symbol of a text in code of X turns out on crossing of a line E and column T. Other part of a source text is ciphered in the similar way.

Source text: ATTACKATDAWN

Key: LEMONLEMONLE

The ciphered text: LXFOPVEFRNHR

Deciphering is made as follows: we find the line corresponding to the first symbol of a keyword in Vizhener's table; we find the first symbol of the ciphered text in this line. The column in which there is this symbol, corresponds to the first symbol of a source text. The following symbols of the ciphered text are deciphered in this way.

If letters A — Z correspond to numbers 0 — 25, Vizhener's enciphering can be written down in the form of a formula:

$$C_i \equiv (P_i + K_i) \mod 26$$

Interpretation:

$$P_i \equiv (C_i - K_i + 26) \mod 26$$

10 Code Vernama

The cryptosystem was offered for enciphering of cable messages which represented binary texts in which the clear text is represented in Baudot's code (in the form of five-digit "pulse combinations"). In this code, for example, the letter "A" had an appearance (1 1 0 0 0). On a paper tape to figure "1" there corresponded the opening, and figure "0" — its absence

For work of a shifrotekst the clear text unites the operation "excluding OR" with the key (called by a disposable notebook or shifrobloknoty). Thus the key has to possess three crucial properties

To have distribution casual evenly: P_ {k} (k) =1/2^ {N}, where k — a key, and N — quantity of binary symbols in a key;

To coincide by the size with the set clear text;

To be applied only once.

It is possible to unite not only bits of the message with key bits, but also, for example, letters. Vernam's idea can be illustrated with attraction of the code of Vizhener. The key for this code, according to Vernam, had to represent casual sequence of letters.

EVTIQWXQVVOPMCXREPYZ key

ALLSWELLTHATENDSWELL clear text

IШифротекст EGEAMAIBOCOIQPAJATJK

Cryptographed message of EGEAM AIBOC OIQPA JATJK

Without knowledge of a key such message doesn't give in to the analysis. Even if it would be possible to try all keys, as result we would receive all possible messages of this length plus enormous number of the senseless decodings looking as a chaotic heap of letters. But also among intelligent decodings there would be

no opportunity to choose the required. When the casual sequence (key) is combined with not casual (in clear), the result of it (шифротекст) is absolutely casual and, therefore, deprived of those statistical features which could be used for the analysis of the code.

11 Codes of programs

Program code Viginer method

```
#include <vcl.h>
#pragma hdrstop
#include <string>
#include <string.h>
#include <math.h>
#include <vector>
#include "Unit2.h"
__fastcall TForm2::TForm2(TComponent* Owner)
      : TForm(Owner)
{
}
void __fastcall TForm2::EncodingClick(TObject *Sender)
{
      char  alf[256];
      int LenA=256;
      int f_str=2, index_k=-1,p,z,ienk=0;
      const int len=Memo1->Text.Length();
      int *is= new int [len];
      int *ik= new int [len];
      char *str=new char[Memo1->Text.Length()];
      String enk="";
      strcpy (str, Memo1->Text.c_str());
      char *key=new char[Memo3->Text.Length()];
      strcpy (key, Memo3->Text.c_str());
      for(int i=0;i<256;i++)
       {alf[i]=(char)i;}
         for(int i=0; i<strlen(str); i++)
          for(int j=0; j<LenA; j++)
           if(str[i]==alf[j])
```

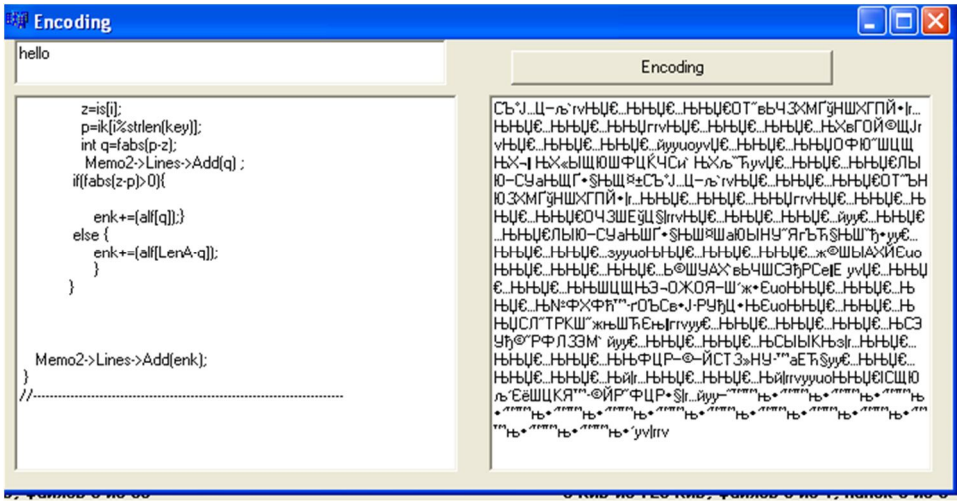
```
            {
             is[i]=j;
            }


          for(int i=0; i<strlen(key); i++)
           for(int j=0; j<LenA; j++)
            if(key[i]==alf[j])
            {
               ik[i]=j;
            }
          for(int i=0; i<strlen(str); i++)
           {
            z=is[i];p=ik[i%strlen(key)];
           if(z+p<LenA){
                  enk+=alf[z+p];}
           else {
              enk+= alf[z+p-LenA];
               }
          }
            Memo2->Lines->Add(enk);
 }
#include <vcl.h>
#pragma hdrstop
#include <string>
#include <string.h>
#include <math.h>
#include "Unit3.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm3 *Form3;
__fastcall TForm3::TForm3(TComponent* Owner)
      : TForm(Owner)
{
}
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    char  alf[256];
     int LenA=26;
```

```
int p,z;
const int len=Memo1->Text.Length();
int *is= new int [len];
int *ik= new int [len];
char *str=new char[Memo1->Text.Length()];
String enk="";
strcpy (str, Memo1->Text.c_str());
char *key=new char[Memo3->Text.Length()];
strcpy (key, Memo3->Text.c_str());

for(int i=97;i<123;i++)
 {alf[i-97]=(char)i;}
   for(int i=0; i<strlen(str); i++)
    for(int j=0; j<LenA; j++)
     if(str[i]==alf[j])
      {
       is[i]=j;
      }
   for(int i=0; i<strlen(key); i++)
    for(int j=0; j<LenA; j++)
     if(key[i]==alf[j])
      {
         ik[i]=j;

      }
   for(int i=0; i<strlen(str); i++)
    {

      z=is[i];
      p=ik[i%strlen(key)];
      int q=fabs(p-z);
       Memo2->Lines->Add(q) ;
     if(fabs(z-p)>0){

         enk+=(alf[q]);}
     else {
         enk+=(alf[LenA-q]);
         }
    }
```

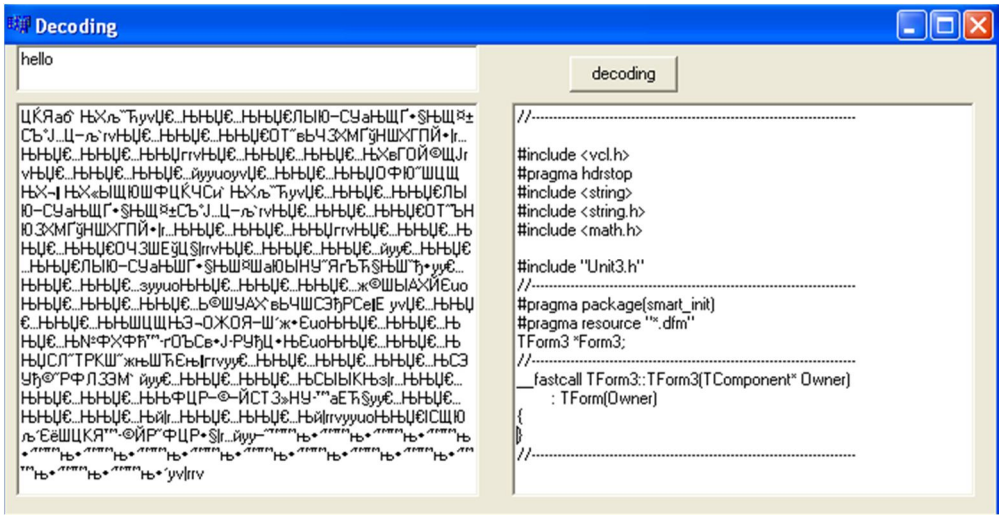
```
        Memo2->Lines->Add(enk);
      }
     //---------------------------------------------------------------------------
```

Results of the program Viginer method

Encryption



Decryption



Program code Vernam

```
#include <vcl.h>
#include <iostream.h>
#include <iomanip>
#include <fstream>
#include<math.h>
```

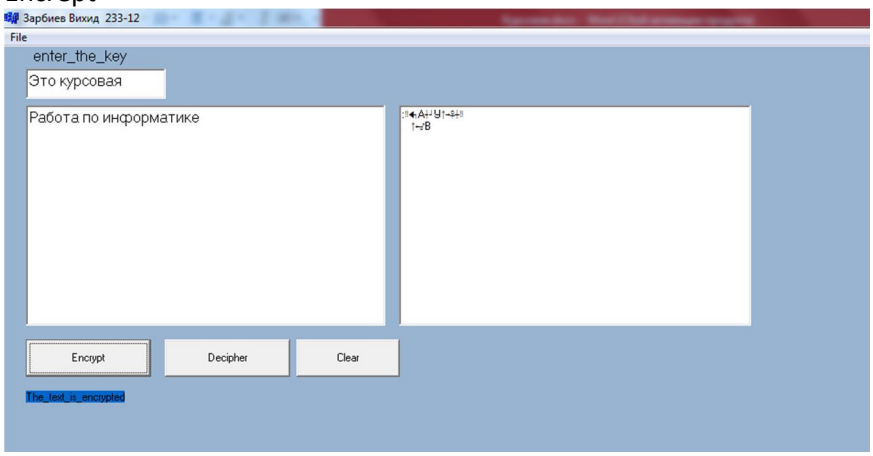
```
#include <cstring.h>
#pragma hdrstop
#include "Unit1.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}

void __fastcall TForm1::enter_the_keyChange(TObject *Sender)
{
  enter_the_key->Lines->SaveToFile( "key.txt" );
}
void __fastcall TForm1::Enter_the_text_to_encryptChange(TObject *Sender)
{
Enter_the_text_to_encrypt->Lines->SaveToFile( "vaxid.txt" );
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    setlocale(LC_ALL, "rus");
    int flag=0;
    string chkey;
    char chtxt;
    char x[1000000],x1;
    int shk=0;
    int k=0;
   // открываем фаил откуда берем текст для шифра
   ifstream fin("vaxid.txt",ios::in|ios::binary);
     while(fin) { // в цикле считываем фаил до конца текста
        fin.get(chtxt);//считваем каждую букву
        char chkey1=chkey[shk];
        x1=chtxt^chkey1;
        x[k]=x1;
        k++;
        shk++;
        if (shk==chkey.size()){shk=0; } //проверяем размер ключа
              flag=1;
               }
      for(int i=0;i<k-1;i++)fout.put(x[i]); // занесем шифрованый текст в фаил 2
    fin.close();
    fout.close();
    if(flag=1)
        {
```

```
        Enter_the_text_for_interpretation->Lines->LoadFromFile("vaxid2.txt");
        The_text_is_encrypted->Visible=true;
         }
}
void __fastcall TForm1::Button2Click(TObject *Sender)
{
//ShellExecute(Handle, "open","2.exe",NULL,NULL,SW_RESTORE);
 setlocale(LC_ALL, "rus");
    int flag=0;
    string chkey;
    char chtxt;
    char x[1000000],x1;
    int shk=0;
    int k=0;
   // открываем фаил откуда берем текст для шифра
   ifstream fin("vaxid2.txt",ios::in|ios::binary);
        while(key) { // в цикле считываем key фаил
    key>>chkey;//считваем каждую букву
    }
     while(fin) { // в цикле считываем фаил до конца текста
        fin.get(chtxt);//считваем каждую букву
        char chkey1=chkey[shk];
        x1=chtxt^chkey1;
        x[k]=x1;
        k++;
        shk++;
        if (shk==chkey.size()){shk=0; } //проверяем размер ключа
              flag=1;
               }
      for(int i=0;i<k-1;i++)fout.put(x[i]); // занесем шифрованый текст в фаил 2

    fin.close();
    fout.close();

    if(flag=1){
        Enter_the_text_to_encrypt->Lines->LoadFromFile("vaxid3.txt");
        Decrypted_text->Visible=true;
        }
}
void __fastcall TForm1::Open11Click(TObject *Sender)
{
if (OpenDialog1->Execute()) Enter_the_text_for_interpretation->Lines-
>LoadFromFile(OpenDialog1->FileName);
}
void __fastcall TForm1::Open21Click(TObject *Sender)
```

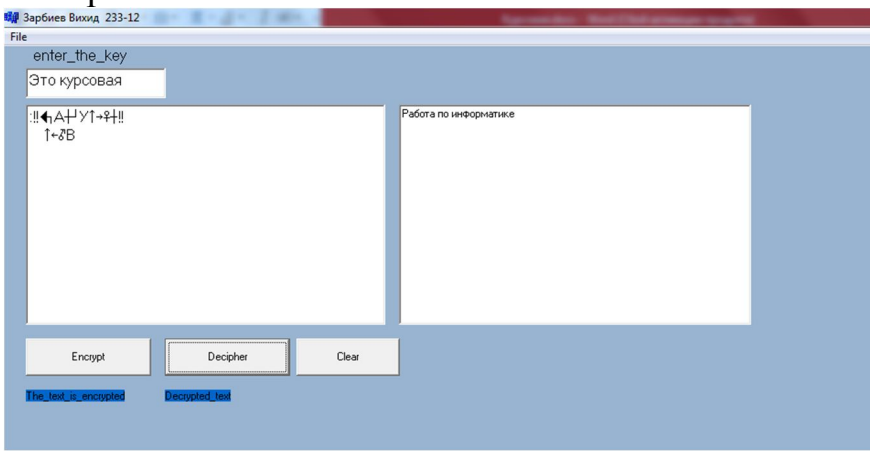
{
if (OpenDialog2->Execute()) Enter_the_text_to_encrypt->Lines->LoadFromFile(OpenDialog2->FileName);

Results of the program Vernam

Encrept



Decript



Program code RSA

```
package rsa;

import java.security.NoSuchAlgorithmException;
import javax.crypto.NoSuchPaddingException;
public class RSA {

    public static void main(String[] args) throws NoSuchAlgorithmException,
NoSuchPaddingException {



    }

}
package rsa;
```

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.math.BigInteger;
import java.security.*;
import java.security.spec.*;
import java.util.logging.*;
import javax.crypto.*;
import javax.swing.*;

public class RSA_1 extends JFrame implements ActionListener{
   public String dir_patch = "";
   JLabel l1,l2,l3;
   JTextArea t1,t2;
   JTextField tf1,tf2;
   JButton b1,b2,b3,b4,b5;
   public RSA_1(){
      setLayout(new BorderLayout());
      setBackground(Color.ORANGE);
      setSize(1000, 200);

      Panel panel = new Panel();
      panel.setLayout(new GridLayout());
      l1 = new JLabel("Plaintext...");
      l2 = new JLabel("Ciphertext...");
      panel.add(l1);
      panel.add(l2);
      this.add(panel, "North");

      Panel centerPanel = new Panel();
      centerPanel.setLayout(new GridLayout());
      t1 = new JTextArea();
      t2 = new JTextArea();
      t2.setBackground(Color.YELLOW);
      centerPanel.add(t1);
      centerPanel.add(t2);
      this.add(centerPanel, "Center");

      Panel southPanel = new Panel();
```

```
    southPanel.setLayout(new GridLayout(4,2));
    b3 = new JButton("Private key ");
    b3.addActionListener(this);
    b4 = new JButton("Public key ");
    b4.addActionListener(this);
    l3 = new JLabel("Press to generate keys: ");
    b5 = new JButton("Generate");
    b5.addActionListener(this);
    tf1 = new JTextField("");
    tf2 = new JTextField("");
    b1 = new JButton("Encrypt");
    b1.addActionListener(this);
    b2 = new JButton("Dencrypt");
    b2.addActionListener(this);
    southPanel.add(b3);
    southPanel.add(b4);
    southPanel.add(l3);
    southPanel.add(b5);
    southPanel.add(tf1);
    southPanel.add(tf2);
    southPanel.add(b1);
    southPanel.add(b2);
    this.add(southPanel, "South");

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
    setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {
    JButton button = (JButton) e.getSource();
    if (button==b1) {try {
            RSAencrypt rsac = new RSAencrypt();
            try {
                try {
                    t2.setText(rsac.RSAencrypt(t1.getText(), tf1.getText()));
                } catch (UnsupportedEncodingException ex) {
```

```
Logger.getLogger(RSA_1.class.getName()).log(Level.CONFIG, null, ex);
                }
            } catch (InvalidKeyException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IllegalBlockSizeException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            } catch (BadPaddingException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            } catch (FileNotFoundException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            } catch (InvalidKeySpecException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            }
        } catch (NoSuchAlgorithmException ex) {
            Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE,
null, ex);
        } catch (NoSuchPaddingException ex) {
            Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE,
null, ex);
        }
}
      else if (button==b2) {try {
            RSAencrypt rsac = new RSAencrypt();
            try {
               try {
                  t2.setText(rsac.RSAdecrypt(t1.getText(), tf2.getText()));
               } catch (UnsupportedEncodingException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
                }
            } catch (InvalidKeyException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
```

```
            } catch (IllegalBlockSizeException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            } catch (BadPaddingException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            } catch (FileNotFoundException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            } catch (InvalidKeySpecException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            }
         } catch (NoSuchAlgorithmException ex) {
            Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE,
null, ex);
         } catch (NoSuchPaddingException ex) {
            Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE,
null, ex);
         }
      }
      else if (button==b3) {getfile();tf1.setText(dir_patch);}
      else if (button==b4) {getfile();tf2.setText(dir_patch);}
      else if (button==b5) {try {
            try {
               generateKeys(tf1.getText(), tf2.getText());
            } catch (FileNotFoundException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IOException ex) {

Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE, null, ex);
            }
         } catch (NoSuchAlgorithmException ex) {
            Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE,
null, ex);
         } catch (InvalidKeySpecException ex) {
            Logger.getLogger(RSA_1.class.getName()).log(Level.SEVERE,
null, ex);
```

```
        }
      }
    }

    public void getfile(){
      JFileChooser fileChooser = new JFileChooser();

fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
      int ret = fileChooser.showDialog(this, "Open");
      String patch = null;
      if (ret == JFileChooser.APPROVE_OPTION)
      {
        patch = fileChooser.getSelectedFile().getAbsolutePath();
        dir_patch = patch;
        final File folder = new File(patch);
      }
    }
    public void generateKeys(String publicFile,String privateFile) throws NoSuchAlgorithmException, InvalidKeySpecException, FileNotFoundException, IOException{
      KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");
      KeyPair keyPair = keyPairGen.generateKeyPair();
PrivateKey privateKey = keyPair.getPrivate();
PublicKey publicKey = keyPair.getPublic();

KeyFactory keyFactory = KeyFactory.getInstance("RSA");
RSAPrivateKeySpec rSAPrivateKey = keyFactory.getKeySpec(privateKey, RSAPrivateKeySpec.class);

BigInteger pr_m = rSAPrivateKey.getModulus();
BigInteger pr_x = rSAPrivateKey.getPrivateExponent();

RSAPublicKeySpec rsaPublicKey = keyFactory.getKeySpec(publicKey, RSAPublicKeySpec.class);

BigInteger pub_m = rsaPublicKey.getModulus();
BigInteger pub_x = rsaPublicKey.getPublicExponent();
```

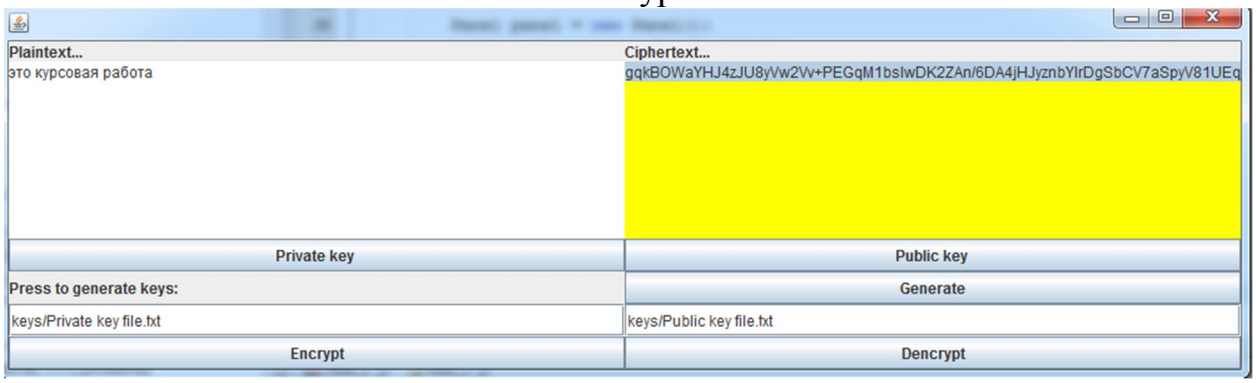
```
String pubf = "keys/"+tf1.getText()+".txt";
String prf = "keys/"+tf2.getText()+".txt";

File publicKeyFile = new File(pubf);
File privateKeyFile = new File(prf);

FileOutputStream f1 = null;
FileOutputStream f2 = null;

f1 = new FileOutputStream(publicKeyFile);
f2 = new FileOutputStream(privateKeyFile);

f1.write(pub_m.toString().getBytes());
f1.write(" ".getBytes());
f1.write(pub_x.toString().getBytes());

f2.write(pr_m.toString().getBytes());
f2.write(" ".getBytes());
f2.write(pr_x.toString().getBytes());

tf1.setText(pubf);
tf2.setText(prf);
   }
}

package rsa;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import java.io.*;
import java.math.BigInteger;
import java.security.*;
import java.security.spec.*;
import java.util.Scanner;
import javax.crypto.*;
class RSAencrypt{
   Cipher c;
   RSAencrypt()  throws  NoSuchAlgorithmException,
NoSuchPaddingException{
```

```
    c = Cipher.getInstance("RSA");
    }
public String RSAencrypt(String plaintext, String file) throws
InvalidKeyException, IllegalBlockSizeException, BadPaddingException,
FileNotFoundException, NoSuchAlgorithmException,
InvalidKeySpecException, UnsupportedEncodingException{
int k = 0;
Scanner reader = new Scanner(new File(file));
String[] str = new String[2];
while (reader.hasNext()){
    str[k] = reader.next();k++;
}

BigInteger pub_m = new BigInteger(str[0]);
BigInteger pub_x = new BigInteger(str[1]);

KeyFactory keyFactory = KeyFactory.getInstance("RSA");
RSAPublicKeySpec new_pubks = new RSAPublicKeySpec(pub_m, pub_x);
PublicKey new_public = keyFactory.generatePublic(new_pubks);

c.init(Cipher.ENCRYPT_MODE, new_public);
byte[] encrypted = c.doFinal(plaintext.getBytes("utf8"));

String result = Base64.encode(encrypted);
return result;
}

public String RSAdecrypt(String ciphertext, String file) throws
FileNotFoundException, NoSuchAlgorithmException,
InvalidKeySpecException, InvalidKeyException,
IllegalBlockSizeException, BadPaddingException,
UnsupportedEncodingException{
int k = 0;
Scanner reader = new Scanner(new File(file));
String[] str = new String[2];
while (reader.hasNext()){
    str[k] = reader.next();k++;
}
```

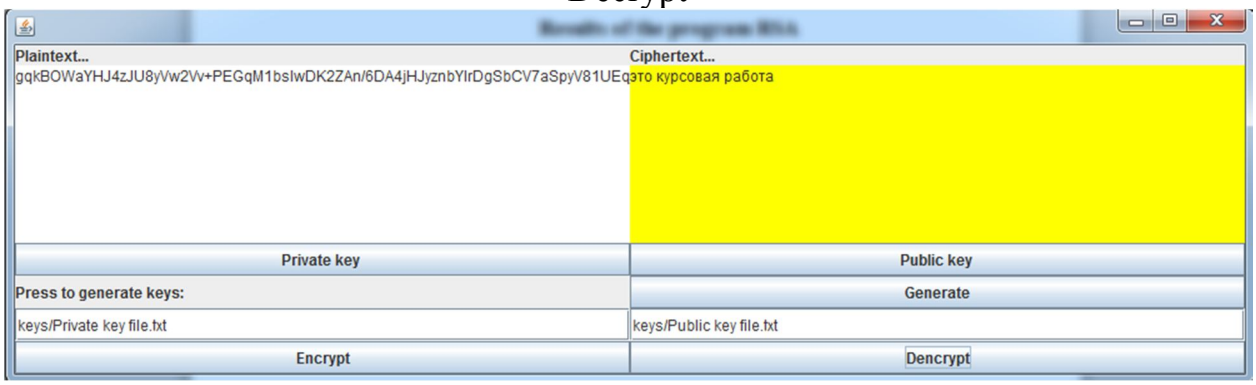
```
BigInteger pr_m = new BigInteger(str[0]);
BigInteger pr_x = new BigInteger(str[1]);

KeyFactory keyFactory = KeyFactory.getInstance("RSA");
RSAPrivateKeySpec new_prks = new RSAPrivateKeySpec(pr_m, pr_x);
PrivateKey new_private = keyFactory.generatePrivate(new_prks);

    byte[] data1=Base64.decode(ciphertext);
    c.init(Cipher.DECRYPT_MODE, new_private);
    byte[] decrypted = c.doFinal(data1);

    String result = new String(decrypted);
    return result;
}
}
```

Results of the program RSA

Encrypt



Decrypt



The list of the used literature

https://ru.wikipedia.org/wiki/Шифр_Вернама
https://ru.wikipedia.org/wiki/Шифр_Виженера
http://en.wikipedia.org/wiki/Euler's_totient_function
https://ru.wikipedia.org/wiki/RSA
http://citforum.ru/book/cryptogr/fullsoder.shtml