ПРИЛОЖЕНИЕ А

Исходный код программы криптографической защиты информации на базе библиотек CryptoPro.Sharpei для работы с СКЗИ «КриптоПро CSP» на платформе программирования .Net

Файл Program.cs

```
using System;
using System.Windows.Forms;

namespace CryptoProTool
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Файл Form1.Designer.cs

```
namespace CryptoProTool
{
    partial class Form1
    {
        private System.ComponentModel.IContainer components = null;
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code
        private void InitializeComponent()
        {
            this.UserName = new System.Windows.Forms.Label();
            this.ContainerLoad = new System.Windows.Forms.Button();
            this.ContainerBox = new System.Windows.Forms.GroupBox();
            this.ContainerExport = new System.Windows.Forms.Button();
            this.ContainerText = new System.Windows.Forms.TextBox();
            this.SignatureBox = new System.Windows.Forms.GroupBox();
            this.SignatureCheck = new System.Windows.Forms.Button();
            this.SignatureSign = new System.Windows.Forms.Button();
            this.CypherBox = new System.Windows.Forms.GroupBox();
            this.CypherDelete = new System.Windows.Forms.Button();
            this.CypherDecrypt = new System.Windows.Forms.Button();
            this.CypherEncrypt = new System.Windows.Forms.Button();
```

```
            this.ContainerBox.SuspendLayout();
            this.SignatureBox.SuspendLayout();
            this.CypherBox.SuspendLayout();
            this.SuspendLayout();
            // UserName
            this.UserName.Cursor = System.Windows.Forms.Cursors.Default;
            this.UserName.Location = new System.Drawing.Point(18, 13);
            this.UserName.Name = "UserName";
            this.UserName.Size = new System.Drawing.Size(124, 20);
            this.UserName.TabIndex = 0;
            this.UserName.Text = "Test text";
            // ContainerLoad
            this.ContainerLoad.Location = new System.Drawing.Point(6,19);
            this.ContainerLoad.Name = "ContainerLoad";
            this.ContainerLoad.Size = new System.Drawing.Size(124, 23);
            this.ContainerLoad.TabIndex = 2;
            this.ContainerLoad.Text = "Загрузить";
            this.ContainerLoad.UseVisualStyleBackColor = true;
            this.ContainerLoad.Click += new
System.EventHandler(this.ContainerLoadClick);
            // ContainerBox
            this.ContainerBox.Controls.Add(this.ContainerExport);
            this.ContainerBox.Controls.Add(this.ContainerLoad);
            this.ContainerBox.Location = new System.Drawing.Point(12,39);
            this.ContainerBox.Name = "ContainerBox";
            this.ContainerBox.Size = new System.Drawing.Size(136, 108);
            this.ContainerBox.TabIndex = 2;
            this.ContainerBox.TabStop = false;
            this.ContainerBox.Text = "Сертификаты";
            // ContainerExport
            this.ContainerExport.Enabled = false;
            this.ContainerExport.Location = new System.Drawing.Point(6,57);
            this.ContainerExport.Name = "ContainerExport";
            this.ContainerExport.Size = new System.Drawing.Size(124, 37);
            this.ContainerExport.TabIndex = 3;
            this.ContainerExport.Text = "Экспортировать открытый ключ";
            this.ContainerExport.UseVisualStyleBackColor = true;
            this.ContainerExport.Click += new
System.EventHandler(this.ContainerExportKeyClick);
            // ContainerText
            this.ContainerText.AcceptsReturn = true;
            this.ContainerText.Location = new System.Drawing.Point(163, 49);
            this.ContainerText.Multiline = true;
            this.ContainerText.Name = "ContainerText";
            this.ContainerText.ReadOnly = true;
            this.ContainerText.ScrollBars =
System.Windows.Forms.ScrollBars.Vertical;
            this.ContainerText.Size = new System.Drawing.Size(286, 296);
            this.ContainerText.TabIndex = 3;
            // SignatureBox
            this.SignatureBox.Controls.Add(this.SignatureCheck);
            this.SignatureBox.Controls.Add(this.SignatureSign);
            this.SignatureBox.Location = new System.Drawing.Point(12, 153);
            this.SignatureBox.Name = "SignatureBox";
            this.SignatureBox.Size = new System.Drawing.Size(136, 85);
            this.SignatureBox.TabIndex = 4;
```

```
            this.SignatureBox.TabStop = false;
            this.SignatureBox.Text = "ЭЦП";
            // SignatureCheck
            this.SignatureCheck.Location = new System.Drawing.Point(6, 50);
            this.SignatureCheck.Name = "SignatureCheck";
            this.SignatureCheck.Size = new System.Drawing.Size(124, 23);
            this.SignatureCheck.TabIndex = 1;
            this.SignatureCheck.Text = "Проверить";
            this.SignatureCheck.UseVisualStyleBackColor = true;
            this.SignatureCheck.Click += new
System.EventHandler(this.SignatureCheckClick);
            // SignatureSign
            this.SignatureSign.Enabled = false;
            this.SignatureSign.Location = new System.Drawing.Point(6, 20);
            this.SignatureSign.Name = "SignatureSign";
            this.SignatureSign.Size = new System.Drawing.Size(124, 23);
            this.SignatureSign.TabIndex = 0;
            this.SignatureSign.Text = "Подписать";
            this.SignatureSign.UseVisualStyleBackColor = true;
            this.SignatureSign.Click += new
System.EventHandler(this.SignatureSignClick);
            // CypherBox
            this.CypherBox.Controls.Add(this.CypherDelete);
            this.CypherBox.Controls.Add(this.CypherDecrypt);
            this.CypherBox.Controls.Add(this.CypherEncrypt);
            this.CypherBox.Location = new System.Drawing.Point(12, 244);
            this.CypherBox.Name = "CypherBox";
            this.CypherBox.Size = new System.Drawing.Size(136, 107);
            this.CypherBox.TabIndex = 5;
            this.CypherBox.TabStop = false;
            this.CypherBox.Text = "Шифрование";
            // CypherDelete
            this.CypherDelete.Location = new System.Drawing.Point(7, 78);
            this.CypherDelete.Name = "CypherDelete";
            this.CypherDelete.Size = new System.Drawing.Size(126, 23);
            this.CypherDelete.TabIndex = 7;
            this.CypherDelete.Text = "Удалить файл";
            this.CypherDelete.UseVisualStyleBackColor = true;
            this.CypherDelete.Click += new
System.EventHandler(this.CypherDeleteClick);
            // CypherDecrypt
            this.CypherDecrypt.Enabled = false;
            this.CypherDecrypt.Location = new System.Drawing.Point(7, 50);
            this.CypherDecrypt.Name = "CypherDecrypt";
            this.CypherDecrypt.Size = new System.Drawing.Size(126, 23);
            this.CypherDecrypt.TabIndex = 1;
            this.CypherDecrypt.Text = "Расшифровать файл";
            this.CypherDecrypt.UseVisualStyleBackColor = true;
            this.CypherDecrypt.Click += new
System.EventHandler(this.CypherDecryptClick);
            // CypherEncrypt
            this.CypherEncrypt.Enabled = false;
            this.CypherEncrypt.Location = new System.Drawing.Point(7, 20);
            this.CypherEncrypt.Name = "CypherEncrypt";
            this.CypherEncrypt.Size = new System.Drawing.Size(126, 23);
            this.CypherEncrypt.TabIndex = 0;
```

```
            this.CypherEncrypt.Text = "Зашифровать файл";
            this.CypherEncrypt.UseVisualStyleBackColor = true;
            this.CypherEncrypt.Click += new
System.EventHandler(this.CypherEncryptClick);
            // Form1
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(467, 362);
            this.Controls.Add(this.ContainerText);
            this.Controls.Add(this.CypherBox);
            this.Controls.Add(this.SignatureBox);
            this.Controls.Add(this.ContainerBox);
            this.Controls.Add(this.UserName);
            this.Name = "Form1";
            this.Text = "Crypto Pro Tool";
            this.ContainerBox.ResumeLayout(false);
            this.SignatureBox.ResumeLayout(false);
            this.CypherBox.ResumeLayout(false);
            this.ResumeLayout(false);
            this.PerformLayout();
        }
        #endregion

        private System.Windows.Forms.Label UserName;
        private System.Windows.Forms.Button ContainerLoad;
        private System.Windows.Forms.GroupBox ContainerBox;
        private System.Windows.Forms.TextBox ContainerText;
        private System.Windows.Forms.GroupBox SignatureBox;
        private System.Windows.Forms.Button SignatureCheck;
        private System.Windows.Forms.Button SignatureSign;
        private System.Windows.Forms.GroupBox CypherBox;
        private System.Windows.Forms.Button CypherDecrypt;
        private System.Windows.Forms.Button CypherEncrypt;
        private System.Windows.Forms.Button CypherDelete;
        private System.Windows.Forms.Button ContainerExport;
    }
}
```

Файл Form1.cs

```
using System;
using System.Security.Cryptography.X509Certificates;
using System.Windows.Forms;
using CryptoPro.Sharpei;
using System.IO;
using System.Security.Cryptography;
using System.Runtime.Serialization.Formatters.Binary;

namespace CryptoProTool
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            UserName.Text = "Загрузите сертификат";
        }
```

```
        private Gost3410CryptoServiceProvider gost;
        private void ContainerLoadClick(object sender, EventArgs e)
        {
            try
            {
                var store = new X509Store();
                store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
                var collection = store.Certificates;
                // Отображаем окно выбора сертификата.
                var scollection =
                    X509Certificate2UI.SelectFromCollection(collection,
                    "Выбор секретного ключа по сертификату",
                   "Выберите сертификат, соответствующий Вашему секретному
ключу.",
                    X509SelectionFlag.SingleSelection);
                // Проверяем, что выбран сертификат
                if (scollection.Count == 0)
                {
                    ContainerTextAppend("Не выбран ни один сертификат.");
                    return;
                }
                // Выбран может быть только один сертификат.
                var certificate = scollection[0];
               // Получаем секретный ключ, соответствующий данному
сертификату.
                if (certificate.HasPrivateKey)
                {
                    gost =
(Gost3410CryptoServiceProvider)certificate.PrivateKey;
                    ContainerTextAppend("Загружен сертификат.");
                    ContainerTextAppend("Теперь Вы можете подписывать,
шифровать и расшифровать файлы от имени субъекта.");
                    ContainerTextAppend("Субъект: " + certificate.Subject);
                    UserName.Text = certificate.Subject;
                    ContainerTextAppend("Алгоритм подписи: " +
certificate.PublicKey.Oid.FriendlyName);
                    EnableCryptoOperations();
                }
                else
                    ContainerTextAppend("Не получается получить секретный
ключ, соответствующий данному сертификату.");
            }
            catch (CryptographicException cE)
            {
                ContainerTextAppend(cE.Message);
            }
        }

        private void ContainerExportKeyClick(object sender, EventArgs e)
        {
            var saveFileDialog = new SaveFileDialog();
            if (saveFileDialog.ShowDialog() == DialogResult.OK &&
saveFileDialog.FileName.Length > 0)
            {
                using (var ofs = new FileStream(saveFileDialog.FileName,
FileMode.Create))
```

```
                {
                    // Передаем открытый ключ.
                    var formatter = new BinaryFormatter();
                    formatter.Serialize(ofs, gost.ExportParameters(false));
                   ContainerTextAppend("Открытый ключ успешно
экспортирован.");
                }
            }
            else
              ContainerTextAppend("Экспортировать открытый ключ не
удалось.");
        }

        private void SignatureSignClick(object sender, EventArgs e)
        {
var openFileDialog = new OpenFileDialog { Title = "Выберите файл для подписи"
};
            if (openFileDialog.ShowDialog() != DialogResult.OK)
            {
                ContainerTextAppend("Файл для подписи не был выбран.");
                return;
            }

            var fs = File.Open(openFileDialog.FileName, FileMode.Open,
FileAccess.Read, FileShare.None);

            var gostHash = new Gost3411CryptoServiceProvider();
            var signedData = gost.SignData(fs, gostHash);
            var saveFileDialog = new SaveFileDialog();
            if (saveFileDialog.ShowDialog() == DialogResult.OK &&
saveFileDialog.FileName.Length > 0)
            {
                using (var ofs = new FileStream(saveFileDialog.FileName,
FileMode.Create))
                {
                    var bw = new BinaryWriter(ofs);
                    bw.Write(signedData.Length);
                    bw.Write(signedData);
                    ContainerTextAppend("Файл успешно подписан.");
                }
            }
            else
                ContainerTextAppend("Файл не был подписан.");
            fs.Close();
        }

        private void SignatureCheckClick(object sender, EventArgs e)
        {
            try
            {
                var openFileDialog = new OpenFileDialog { Title = "Выберите
подписанный файл" };
                if (openFileDialog.ShowDialog() != DialogResult.OK)
                {
                    ContainerTextAppend("Подписанный файл не был выбран.");
                    return;
```

```
                }
                var rawData = File.ReadAllBytes(openFileDialog.FileName);
                var openSignatureDialog = new OpenFileDialog { Title =
"Выберите файл подписи" };
                if (openSignatureDialog.ShowDialog() != DialogResult.OK)
                {
                    ContainerTextAppend("Файл с подписью не был выбран.");
                    return;
                }
                var openPublicKeyDialog = new OpenFileDialog { Title =
"Выберите файл с открытым ключом подписавшего" };
                if (openPublicKeyDialog.ShowDialog() != DialogResult.OK)
                {
                  ContainerTextAppend("Файл с открытым ключом не был
выбран.");
                    return;
                }

                using (var ifs = new FileStream(openSignatureDialog.FileName,
FileMode.Open, FileAccess.Read))
                {
                    var alg = new Gost3410CryptoServiceProvider();
                    using (var ifsPK = new
FileStream(openPublicKeyDialog.FileName, FileMode.Open, FileAccess.Read))
                    {
                        //читаем публичный ключ подписавшего
                        var formatterPK = new BinaryFormatter();
  alg.ImportParameters((Gost3410Parameters)formatterPK.Deserialize(ifsPK));
                    }
                    var br = new BinaryReader(ifs);
                    //читаем подпись
                    var signatureLength = br.ReadInt32();
                    var signature = br.ReadBytes(signatureLength);
                    var gostHash = new Gost3411CryptoServiceProvider();
             ContainerTextAppend(alg.VerifyData(rawData, gostHash, signature)
?
                        "Подпись верна" : "Подпись не верна");
                }
            }
            catch (Exception ex)
            {
                ContainerTextAppend("Не удалось проверить подпись: " +
ex.Message);
            }
        }

        private void CypherEncryptClick(object sender, EventArgs e)
        {
            var store = new X509Store("MY", StoreLocation.CurrentUser);
            store.Open(OpenFlags.OpenExistingOnly | OpenFlags.ReadWrite);
            var fcollection = store.Certificates;
            var collection =
                X509Certificate2UI.SelectFromCollection(fcollection,
                "Выберите сертификат",
                "Выберите сертификат получателя",
                X509SelectionFlag.MultiSelection);
```

```
            if (collection.Count == 0)
            {
                ContainerTextAppend("Сертификат не был выбран.");
                return;
            }
            //выбираем файл для шифрования
            var openFileDialog = new OpenFileDialog { Title = "Выберите файл
для шифрования" };
            if (openFileDialog.ShowDialog() != DialogResult.OK)
            {
                ContainerTextAppend("Файл для шифрования не был выбран.");
                return;
            }
            //создаём симметричный ключ
            var symmetric = new Gost28147CryptoServiceProvider();
            symmetric.GenerateKey();
            symmetric.GenerateIV();
            var encryptor = symmetric.CreateEncryptor();

            //сохраняем зашифрованный файл
            var saveFileDialog = new SaveFileDialog();
            if (saveFileDialog.ShowDialog() != DialogResult.OK ||
saveFileDialog.FileName.Length <= 0)
                return;
            using (var ofs = new FileStream(saveFileDialog.FileName,
FileMode.Create))
            {
                var bw = new BinaryWriter(ofs);
                bw.Write(collection.Count);
                foreach (var cert in collection)
                {
                    // Разбираем сертификат получателя
                    // Открытый ключ получателя.
                    var pk = cert.PublicKey.Key;
                    var alg = pk as Gost3410;
                    if (alg == null)
                   throw new CryptographicException("Not a gost
certificate");
                    // Создаем ключ согласования
                    var agree =
gost.CreateAgree(alg.ExportParameters(false));
                    // Зашифровываем симметричный ключ на ключе согласования.
                    var wrappedKey = agree.Wrap(symmetric,
                        GostKeyWrapMethod.CryptoProKeyWrap);
                    bw.Write(cert.GetSerialNumberString());
                    // Записываем зашифрованный симметричный ключ.
                    bw.Write(wrappedKey.Length);
                    bw.Write(wrappedKey);
                }
                // Записываем синхропосылку
                bw.Write(symmetric.IV.Length);
                bw.Write(symmetric.IV);
                // Передаем открытый ключ.
                var formatter = new BinaryFormatter();
                formatter.Serialize(ofs, gost.ExportParameters(false));
                // Создаем поток шифрования для записи в файл.
```

```
                using (var cs = new CryptoStream(ofs, encryptor,
CryptoStreamMode.Write))
                {
                    var data = new byte[4096];
                    // Открываем входной файл.
                    using (var ifs = new FileStream(openFileDialog.FileName,
FileMode.Open, FileAccess.Read))
                    {
                        // и переписываем содержимое в выходной поток.
                        var length = ifs.Read(data, 0, data.Length);
                        while (length > 0)
                        {
                            cs.Write(data, 0, length);
                            length = ifs.Read(data, 0, data.Length);
                        }
                    }
                }
            }
            ContainerTextAppend("Файл успешно зашифрован.");
        }

        private void CypherDecryptClick(object sender, EventArgs e)
        {
            try
            {
                var dialog = new OpenFileDialog
                {
                    Filter = "Зашифрованные файлы|*.enc|Все файлы|*.*",
                    Title = "Выберите зашифрованный файл"
                };

                if (dialog.ShowDialog() != DialogResult.OK)
                {
                  ContainerTextAppend("Файл для расшифрования не был
выбран.");
                    return;
                }

                using (var ifs = new FileStream(dialog.FileName,
FileMode.Open, FileAccess.Read))
                {
                    // Читаем зашифрованный симметричный ключ.
                    var br = new BinaryReader(ifs);
                    var amount = br.ReadInt32();
                    var wrappedKey = new byte[] { };
                    for (var i = 0; i < amount; i++)
                    {
                        var serial = br.ReadString();
                        // Читаем зашифрованный симметричный ключ.
                        var wrappedKeyLen = br.ReadInt32();
                        var myWrappedKey = br.ReadBytes(wrappedKeyLen);
                        if
(serial.Equals(gost.ContainerCertificate.GetSerialNumberString()))
                        {
                            wrappedKey = myWrappedKey;
                        }
```

```
                    }
                    // Читаем синхропосылку
                    var ivLength = br.ReadInt32();
                    var iv = br.ReadBytes(ivLength);
                    // Читаем открытый ключ.
                    var formatter = new BinaryFormatter();
                    var srcPublicKeyParameters =
                        (Gost3410Parameters)formatter.Deserialize(ifs);
                    // Создаем ключ согласования
                    var agree = gost.CreateAgree(srcPublicKeyParameters);
                    // Расшифровываем симметричный ключ на ключе согласования
                    var symmetric = agree.Unwrap(wrappedKey,
GostKeyWrapMethod.CryptoProKeyWrap);
                    symmetric.IV = iv;
                    // Создаем поток расшифрования.
                    var transform = symmetric.CreateDecryptor();
                    // Создаем поток расшифрования из файла.
                    using (var cs = new CryptoStream(ifs, transform,
CryptoStreamMode.Read))
                    {
                        // Открываем расшифрованный файл
                        var saveFileDialog = new SaveFileDialog();
                        if (saveFileDialog.ShowDialog() != DialogResult.OK ||
saveFileDialog.FileName.Length <= 0)
                            return;
                        using (var ofs = new
FileStream(saveFileDialog.FileName, FileMode.Create))
                        {
                            var data = new byte[4096];
                            // и переписываем содержимое в выходной поток.
                            var length = cs.Read(data, 0, data.Length);
                            while (length > 0)
                            {
                                ofs.Write(data, 0, length);
                                length = cs.Read(data, 0, data.Length);
                            }
                        }
                    }
                    ContainerTextAppend("Файл был успешно расшифрован.");
                }
            }
            catch (Exception cE)
            {
            ContainerTextAppend("Не удалось расшифровать файл: " +
cE.Message);
            }
        }

        private void CypherDeleteClick(object sender, EventArgs e)
        {
            var dialog = new OpenFileDialog
            {
                Filter = "Все файлы|*.*",
                Title = "Выберите файл для гарантированного удаления"
            };
            if (dialog.ShowDialog() != DialogResult.OK)
```

```
            {
                ContainerTextAppend("Файл для удаления не был выбран.");
                return;
            }
            var fi = new FileInfo(dialog.FileName);
            var size = fi.Length;
            using (var ofs = new FileStream(dialog.FileName,
FileMode.Truncate))
            {
                var rng = new RNGCryptoServiceProvider(new CspParameters {
KeyContainerName = UserName.Text, ProviderType = 75 });
                 for (var i = 0; i < size; i += 4096)
                 {
                     var len = size - i < 4096 ? size-i : 4096;
                   var rand = new byte[len];
                   rng.GetBytes(rand);
                   ofs.Write(rand, 0, (int) len);
                }
            }
            const string message = "Файл перезаписан случайными данными.
Удалить?";
            const string caption = "Гарантированное удаление файла";
            var result = MessageBox.Show(message, caption,
                                         MessageBoxButtons.YesNo,
                                         MessageBoxIcon.Question);
            if (result == DialogResult.Yes)
            {
                File.Delete(dialog.FileName);
                ContainerTextAppend("Файл был успешно удалён.");
            }
        }
//вспомогательные функции
        private void ContainerTextAppend(string str)
        {
            ContainerText.AppendText(str + Environment.NewLine);
        }

        private void EnableCryptoOperations()
        {
            ContainerExport.Enabled = true;
            SignatureSign.Enabled = true;
            CypherDecrypt.Enabled = true;
            CypherEncrypt.Enabled = true;
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

Исходный код программы криптографической защиты информации на базе библиотек «Верба-OW»

Verba.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;

namespace VerbaClient
{
    public struct Spr_List
    {
        [MarshalAs(UnmanagedType.ByValArray,SizeConst=13)]
        public char[] key_id;
        public char key_type;
        public char key_status;
        public string KeyID
        {get
            {
                StringBuilder sb = new StringBuilder();
                foreach (char c in key_id)
                {
                    if (c == '\0')
                    {
                        break;
                    }
                    sb.Append(c);
                }
                return sb.ToString();
            }
        }
    }

    public struct Check_Status
    {
        [MarshalAs(UnmanagedType.ByValArray, SizeConst = 13)]
        public char[] name;
        [MarshalAs(UnmanagedType.ByValArray, SizeConst = 121)]
        public char[] alias;
        public byte Position;
        public byte Status;
        public UInt32 Date;
        public string Name
        {get
            {
                StringBuilder sb = new StringBuilder();
                foreach (char c in name)
                {
                    if (c == '\0')
                    {
                        break;
```

```
                }
                sb.Append(c);
            }
            return sb.ToString();
        }
    }

    public string Alias
    {get
        {
            StringBuilder sb = new StringBuilder();
            foreach (char c in alias)
            {
                if (c == '\0')
                {
                    break;
                }
                sb.Append(c);
            }
            return sb.ToString();
        }
    }
}


public class verba
{
    public static void InitRandomNumbers()
    {
        System.Diagnostics.Process p = new System.Diagnostics.Process();
        p.StartInfo = new
System.Diagnostics.ProcessStartInfo("c:\\windows\\asrkeyw.exe", "s");
        p.Start();
        p.WaitForExit();
    }

    public static void ShowLoadedKeys()
    {
        System.Diagnostics.Process p = new System.Diagnostics.Process();
        p.StartInfo = new
System.Diagnostics.ProcessStartInfo("c:\\windows\\asrkeyw.exe");
        p.Start();
        p.WaitForExit();
    }

    private static char[] ConvertStringToNullTerminated(String input)
    {
        List<char> lst;
        lst = input.ToList<char>();
        lst.Add('\0');
        return lst.ToArray();
    }

#region DLL Imports
    [DllImport("wbotho.dll")]
    private static extern UInt16 InitKey(char[] Key_Dev, char[] Key_ID);
```

```
        [DllImport("wbotho.dll")]
        private static extern UInt16 SprList(char[] base_dir, char[] ser,
            [MarshalAs(UnmanagedType.LPArray, SizeParamIndex=3)]
            ref Spr_List[] list,
            ref UInt16 num, char S_or_E);
        [DllImport("wbotho.dll")]
        private static extern UInt16 AddOpenKey(char[] base_dir, byte[]
open_key, char[] my_ID, char S_or_E);
        [DllImport("wbotho.dll")]
        private static extern UInt16 DelOpenKey (char[] base_dir,char[]
open_key_ID, char[] my_ID, char S_or_E);
        [DllImport("wbotho.dll")]
        private static extern UInt16 CryptoInit (char[] pathToSecret, char[]
pathToBase);
        [DllImport("wbotho.dll")]
        private static extern UInt16 CryptoDone ();
        [DllImport("wbotho.dll")]
        private static extern UInt16 EnCryptFile(char[] file_in, char[]
file_out, UInt16 node_From, UInt16[] node_To, char[] ser);
        [DllImport("wbotho.dll")]
        private static extern UInt16 DeCryptFile(char[] file_in, char[]
file_out, UInt16 abonent);
        [DllImport("wbotho.dll")]
        private static extern UInt16 SignInit(char[] pathToSecret, char[]
pathToBase);
        [DllImport("wbotho.dll")]
        private static extern UInt16 SignDone();
        [DllImport("wbotho.dll")]
        private static extern UInt16 SignFile(char[] src_file_name, char[]
dst_file_name, char[] name);
        [DllImport("wbotho.dll")]
        private static extern UInt16 check_file_sign(char[] file_name, ref
byte count, ref Check_Status[] stat_array);
        [DllImport("wbotho.dll")]
        private static extern UInt16 DelSign(char[] file_name, byte count);
#endregion
#region Public Interface
        public static void pInitKey(String Key_Dev)
        {
            UInt16 res;
            res = InitKey(ConvertStringToNullTerminated(Key_Dev),
ConvertStringToNullTerminated(""));
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }

        public static List<Spr_List> pSprList(String base_dir, String ser,
char S_or_E)
        {
            List<Spr_List> list = new List<Spr_List>();
            UInt16 num = 0;
            Spr_List[] arr={};
            UInt16 res;
```

```
            res = SprList(ConvertStringToNullTerminated(base_dir),
ConvertStringToNullTerminated(ser),
                ref arr, ref num, S_or_E);
            if (res > 0)
            {
                throw new verbaException(res);
            }
            for (int i = 0; i < num; i++)
            {
                list.Add(arr[i]);
            }
            return list;
        }

        public static void pDelOpenKey(String base_dir, String open_key_ID,
String my_ID, char S_or_E)
        {
            UInt16 res;
            res = DelOpenKey(ConvertStringToNullTerminated(base_dir),
ConvertStringToNullTerminated(open_key_ID),
                ConvertStringToNullTerminated(my_ID), S_or_E);
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }

        public static void pAddOpenKey(String base_dir, byte[] open_key,
String my_ID, char S_or_E)
        {
            UInt16 res;
            res = AddOpenKey(ConvertStringToNullTerminated(base_dir),
open_key,
                ConvertStringToNullTerminated(my_ID), S_or_E);
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }

        public static void pCryptoInit(String pathToSecret, String
pathToBase)
        {
            UInt16 res;
            res = CryptoInit(ConvertStringToNullTerminated(pathToSecret),
ConvertStringToNullTerminated(pathToBase));
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }

        public static void pCryptoDone()
        {
            UInt16 res;
            res = CryptoDone();
```

```
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }

        public static void pEnCryptFile(String file_in, String file_out,
UInt16 node_From, UInt16[] node_To, String ser)
        {
            UInt16 res;
            res = EnCryptFile(ConvertStringToNullTerminated(file_in),
ConvertStringToNullTerminated(file_out),
                node_From, node_To, ConvertStringToNullTerminated(ser));
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }

        public static void pDeCryptFile(String file_in, String file_out,
UInt16 abonent)
        {
            UInt16 res;
            res = DeCryptFile(ConvertStringToNullTerminated(file_in),
ConvertStringToNullTerminated(file_out), abonent);
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }

        public static void pSignInit(String pathToSecret, String pathToBase)
        {
            UInt16 res;
            res = SignInit(ConvertStringToNullTerminated(pathToSecret),
ConvertStringToNullTerminated(pathToBase));
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }

        public static void pSignDone()
        {
            UInt16 res;
            res = SignDone();
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }

        public static void pSignFile(String src_file_name, String
dst_file_name, String name)
        {
            UInt16 res;
```

```
            res = SignFile(ConvertStringToNullTerminated(src_file_name),
                ConvertStringToNullTerminated(dst_file_name),
ConvertStringToNullTerminated(name));
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }

        public static List<Check_Status> pcheck_file_sign(String file_name)
        {
            List<Check_Status> result = new List<Check_Status>();
            byte count = 0;
            Check_Status[] statuses = {};

            UInt16 res;
            res = check_file_sign(ConvertStringToNullTerminated(file_name),
ref count, ref statuses);
            if (res > 0)
            {
                throw new verbaException(res);
            }
            for (int i = 0; i < count; i++)
            {
                result.Add(statuses[i]);
            }
            return result;
        }

        public static void pDelSign(String fileName)
        {
            UInt16 res;
            res = DelSign(ConvertStringToNullTerminated(fileName), 255);
            if (res > 0)
            {
                throw new verbaException(res);
            }
        }
#endregion
    }
    public class verbaException : Exception
    {
        private UInt16 m_ErrorCode;

        public verbaException(UInt16 ErrorCode)
        {
            m_ErrorCode = ErrorCode;
        }

        public override string Message
        {
            get
            {
                return String.Format("Error in verba lib: {0}", m_ErrorCode);
            }
        }
```

```
    }
}
```

EnterKey.cs (windows form)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace VerbaClient
{
    public partial class EnterKey : Form
    {
        public EnterKey()
        {
            InitializeComponent();
        }

        public String Value;

        private void btnOK_Click(object sender, EventArgs e)
        {
            Value = tbKey.Text;
            this.Close();
        }
    }
}
```

MainForm.cs (windows form)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace VerbaClient
{
    public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            verba.InitRandomNumbers();
        }

        private void btnOpenSecret_Click(object sender, EventArgs e)
```

```
        {
            fbdSelectDir.ShowDialog();
            tbSecretPath.Text = fbdSelectDir.SelectedPath;
        }

        private void btnOpenDictionary_Click(object sender, EventArgs e)
        {
            fbdSelectDir.ShowDialog();
            tbDictionaryPath.Text = fbdSelectDir.SelectedPath;
        }

        private void btnLoadKey_Click(object sender, EventArgs e)
        {try
            {
                verba.pInitKey(tbSecretPath.Text);
                MessageBox.Show("Key loaded successfully");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void btnUnloadKey_Click(object sender, EventArgs e)
        {
            verba.ShowLoadedKeys();
        }

        private void btnEncrypt_Click(object sender, EventArgs e)
        {
            sfdSaveFile.ShowDialog();
            String outFileName = sfdSaveFile.FileName;
            try
            {
                verba.pCryptoInit(tbSecretPath.Text, tbDictionaryPath.Text);
                List<UInt16> lstTo = new List<UInt16>();
                char[] separators = {',', ' ', '.'};
                foreach (String s in tbTo.Text.Split(separators))
                {
                    lstTo.Add(UInt16.Parse(s));
                }
                lstTo.Add(0);

                verba.pEnCryptFile(tbEncrypt.Text, outFileName,
UInt16.Parse(tbFrom.Text), lstTo.ToArray(), tbSeria.Text);
                MessageBox.Show("File encrypted successfully");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
            finally
            {
                verba.pCryptoDone();
            }
        }
```

```
        private void btnDecrypt_Click(object sender, EventArgs e)
        {
            sfdSaveFile.ShowDialog();
            String outFileName = sfdSaveFile.FileName;
            try
            {
                verba.pCryptoInit(tbSecretPath.Text, tbDictionaryPath.Text);
                verba.pDeCryptFile(tbDecrypt.Text, outFileName,
UInt16.Parse(tbDecryptAbonent.Text));
                MessageBox.Show("File decrypted successfully");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
            finally
            {
                verba.pCryptoDone();
            }
        }

        private void btnSelectEncrypt_Click(object sender, EventArgs e)
        {
            ofdSelectFile.ShowDialog();
            tbEncrypt.Text = ofdSelectFile.FileName;
        }

        private void btnSelectDecrypt_Click(object sender, EventArgs e)
        {
            ofdSelectFile.ShowDialog();
            tbDecrypt.Text = ofdSelectFile.FileName;
        }

        private void btnSelectSign_Click(object sender, EventArgs e)
        {
            ofdSelectFile.ShowDialog();
            tbSign.Text = ofdSelectFile.FileName;
        }

        private void btnSelectUnSign_Click(object sender, EventArgs e)
        {
            ofdSelectFile.ShowDialog();
            tbVerify.Text = ofdSelectFile.FileName;
        }

        private void btnSign_Click(object sender, EventArgs e)
        {
            sfdSaveFile.ShowDialog();
            String outFileName = sfdSaveFile.FileName;
            try
            {
                verba.pSignInit(tbSecretPath.Text, tbDictionaryPath.Text);
                verba.pSignFile(tbSign.Text, outFileName, tbSignKey.Text);
                MessageBox.Show("File signed successfully");
            }
```

```
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
            finally
            {
                verba.pSignDone();
            }
        }

        private void btnVerify_Click(object sender, EventArgs e)
        {try
            {
                List<Check_Status> statuses;
                verba.pSignInit(tbSecretPath.Text, tbDictionaryPath.Text);
                statuses = verba.pcheck_file_sign(tbVerify.Text);
                String msg = "";
                foreach (Check_Status cs in statuses)
                {
                    msg += String.Format("Sign {0}: {1}\n", cs.Name,
                        cs.Status == 0 ? "correct" : (cs.Status == 1 ?
"wrong" : "key not found"));

                }
                MessageBox.Show(msg);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
            finally
            {
                verba.pSignDone();
            }
        }

        private void btnUnSign_Click(object sender, EventArgs e)
        {try
            {
                verba.pSignInit(tbSecretPath.Text, tbDictionaryPath.Text);
                verba.pDelSign(tbVerify.Text);
                MessageBox.Show("Signs were removed");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
            finally
            {
                verba.pSignDone();
            }
        }

        private void btnLoadE_Click(object sender, EventArgs e)
        {try
            {
```

```
                lstE.Items.Clear();
                List<Spr_List> list = verba.pSprList(tbDictionaryPath.Text,
tbSeria.Text, 'E');
                foreach (Spr_List item in list)
                {
                    lstE.Items.Add(item.KeyID);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void btnLoadS_Click(object sender, EventArgs e)
        {try
            {
                lstS.Items.Clear();
                List<Spr_List> list = verba.pSprList(tbDictionaryPath.Text,
tbSeria.Text, 'S');
                foreach (Spr_List item in list)
                {
                    lstS.Items.Add(item.KeyID);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void btnAddE_Click(object sender, EventArgs e)
        {try
            {
                EnterKey frm = new EnterKey();
                frm.ShowDialog();
                ofdSelectFile.ShowDialog();
                byte[] fileData =
System.IO.File.ReadAllBytes(ofdSelectFile.FileName);
             verba.pAddOpenKey(tbDictionaryPath.Text,fileData ,frm.Value,
'E');
                lstE.Items.Clear();
                btnLoadE_Click(null, null);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void btnAddS_Click(object sender, EventArgs e)
        {try
            {
                EnterKey frm = new EnterKey();
                frm.ShowDialog();
                ofdSelectFile.ShowDialog();
```

```
                byte[] fileData =
System.IO.File.ReadAllBytes(ofdSelectFile.FileName);
            verba.pAddOpenKey(tbDictionaryPath.Text, fileData, frm.Value,
'S');
                lstS.Items.Clear();
                btnLoadS_Click(null, null);
            }
            catch (Exception ex)
            {  MessageBox.Show(ex.Message);   }
        }

        private void btnDeleteE_Click(object sender, EventArgs e)
        {try
            {
                if (lstE.SelectedItems.Count > 0)
                {
                    EnterKey frm = new EnterKey();
                    frm.ShowDialog();
                    verba.pDelOpenKey(tbDictionaryPath.Text,
lstE.SelectedItems[0].ToString(), frm.Value, 'E');
                    lstE.Items.Clear();
                    btnLoadE_Click(null, null);
                }
            }
            catch (Exception ex)
            { MessageBox.Show(ex.Message); }
        }

        private void btnDeleteS_Click(object sender, EventArgs e)
        { try
            {
                if (lstS.SelectedItems.Count > 0)
                {
                    EnterKey frm = new EnterKey();
                    frm.ShowDialog();
                    verba.pDelOpenKey(tbDictionaryPath.Text,
lstS.SelectedItems[0].ToString(), frm.Value, 'S');
                    lstS.Items.Clear();
                    btnLoadS_Click(null, null);
                }
            }
            catch (Exception ex)
            { MessageBox.Show(ex.Message); }
        }
    }
}
```

ПРИЛОЖЕНИЕ В

Исходный код программы защиты информации от несанкционированного распространения на базе ключей Guardant

```
Imports Guardant
Imports System.Security.Principal

Public Class MainForm

    Private publikKey As UInt32 = 2791255268
    Private privateRead As UInt32 = 4096767457
    Private privateWrite As UInt32 = 3534032523
    Private privateMaster As UInt32 = 2738231312
    Private findInfo As pFindInfo
    Private h As Handle
    Private correctUser As Boolean = False
    Private Sub InitForm()
        CheckUser()
        btnRegister.Enabled = Not correctUser
        btnEncode.Enabled = correctUser
        btnDecode.Enabled = correctUser
    End Sub

    Private Sub MainForm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        initKey()
        InitForm()
    End Sub

    Private Sub MainForm_FormClosing(ByVal sender As System.Object, ByVal e
As System.Windows.Forms.FormClosingEventArgs) Handles MyBase.FormClosing
        deInitKey()
    End Sub

    Private Sub initKey()
        Dim result As GrdE
        Dim keyID As UInt32
        result = Api32.GrdStartup(GrdFMR.Local)
        h = Api32.GrdCreateHandle(GrdCHM.SingleThread)
        result = Api32.GrdSetAccessCodes(h, publikKey, privateRead,
privateWrite, privateMaster)
        result = Api32.GrdSetFindMode(h, GrdFMR.Local, GrdFM.ID, 0,
617180116[1], 0, 0, 0, GrdDT.Win, GrdFMM.GS3U, GrdFMI.USB)
        result = Api32.GrdFind(h, GrdF.First, keyID, findInfo)
        result = Api32.GrdLogin(h, GrdLM.All)
    End Sub

    Private Sub deInitKey()
        Dim result As GrdE
        result = Api32.GrdLogout(h)
        result = Api32.GrdCloseHandle(h)
        result = Api32.GrdCleanup()
    End Sub
```

[1] Следует указать номер ключа, с которым ведется работа

```
    Private Sub CheckUser()
        Dim currentUserHash = GetCurrentUserSIDHash()
        Dim registeredUserHash = GetSIDfromKey()
        'сравнить хэши
        Dim blnCorrectUser As Boolean = True
        For i = 0 To 7
            If currentUserHash(i) <> registeredUserHash(i) Then
                blnCorrectUser = False
            End If
        Next
        correctUser = blnCorrectUser
    End Sub

    Private Sub btnRegister_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRegister.Click
        Dim result As GrdE
        'проверить, не зарегистрирован ли уже кто-то
        Dim registeredUserHash = GetSIDfromKey()
        Dim blnAlreadyRegistered As Boolean = False
        For Each b In registeredUserHash
            If b <> 0 Then
                blnAlreadyRegistered = True
            End If
        Next
        If blnAlreadyRegistered Then
            MsgBox("Для этого ключа уже зарегистрирован другой пользователь")
        Else
            result = Api32.GrdWrite(h, CUInt(246), 8,
GetCurrentUserSIDHash())
            MsgBox("Пользователь успешно зарегистрирован")
            InitForm()
        End If
    End Sub

    Private Function GetCurrentUserSIDHash() As Byte()
        Dim result As GrdE
        'получить SID текущего пользователя
        Dim currentUser As WindowsIdentity
        currentUser = WindowsIdentity.GetCurrent()
        Dim sid(currentUser.User.BinaryLength - 1) As Byte
        currentUser.User.GetBinaryForm(sid, 0)
        'посчитать хэш-значение от SID при помощи аппаратного алгоритма
        Dim currentUserHash(GrdHASH64.DIGEST_SIZE - 1) As Byte
        result = Api32.GrdHash(h, GrdAN.HASH64, sid, GrdSC.All,
currentUserHash)
        Return currentUserHash
    End Function

    Private Function GetSIDfromKey() As Byte()
        Dim result As GrdE
       'считать записанное хэш-значение пользователя
        Dim registeredUserHash(7) As Byte
        result = Api32.GrdRead(h, CUInt(246), CUInt(8), registeredUserHash)
        Return registeredUserHash
    End Function
```

```
    Private Sub btnEncode_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnEncode.Click
        Dim result As GrdE

        If ofd.ShowDialog() Then
            'читаем выбранный файл
            Dim data As Byte() = IO.File.ReadAllBytes(ofd.FileName)
            Dim dataLen = data.Length
            If dataLen Mod 8 <> 0 Then
                'увеличиваем размер массива до значения, кратного 8
                dataLen = dataLen + (8 - dataLen Mod 8)
                ReDim Preserve data(dataLen - 1)
            End If
            'кодируем файл
            result = Api32.GrdCrypt(h, CUInt(0), data, GrdAM.CFB Or
GrdAM.Encode Or GrdSC.All, CLng(56))
            If result = GrdE.OK Then
                'записываем результат в выходной файл и сообщаем об этом
пользователю
                IO.File.WriteAllBytes(ofd.FileName + ".enc", data)
                MsgBox("Файл успешно зашифрован")
            End If
        End If
    End Sub

    Private Sub btnDecode_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnDecode.Click
        Dim result As GrdE

        If ofd.ShowDialog() Then
            Dim data As Byte() = IO.File.ReadAllBytes(ofd.FileName)
            Dim dataLen = data.Length
            If dataLen Mod 8 <> 0 Then
                dataLen = dataLen + (8 - dataLen Mod 8)
            End If
            ReDim Preserve data(dataLen - 1)
            result = Api32.GrdCrypt(h, CUInt(0), data, GrdAM.CFB Or
GrdAM.Decode Or GrdSC.All, CLng(56))
            If result = GrdE.OK Then
                IO.File.WriteAllBytes(ofd.FileName + ".dec", data)
                MsgBox("Файл успешно расшифрован")
            End If
        End If
    End Sub
End Class
```