ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение высшего профессионального образования
«Владимирский государственный университет»
Кафедра конструирования и технологии радиоэлектронных средств»

Методические указания по выполнению лабораторных работ по курсу

Аппаратные и программные средства защиты информации

ВЛАДИМИР 2007

Содержание Стр.

Лабораторная работа N1

Шифрование данных симметричным алгоритмом

Цель работы: получить навыки по использованию симметричных криптографических алгоритмов для шифрования данных.

1. Основные положения.

Существует два основных типа криптографических алгоритмов:
- симметричные, для которых ключ расшифрования совпадает с ключом зашифрования
- асимметричные (алгоритмы с открытым ключом), использующие для зашифрования и расшифрования два разных ключа.

Симметричные алгоритмы делятся на две категории:
- потоковые шифры, в которых данные обрабатываются побитово (посимвольно),
- блочные шифры, в которых операции производятся над группами битов.

Криптостойкость – характеристика шифра, определяющая его стойкость к дешифрованию без знания ключа (основные характеристики: 1. количество всех возможных ключей 2. среднее время, необходимое для криптоанализа).

Общепринятые требования к криптографическим алгоритмам:
1. зашифрованный текст читается только при наличии ключа,
2. число операций для нахождения ключа по фрагменту шифрованного текста и соответствующего ему открытого текста – не менее общего числа возможных ключей
3. число операций для дешифровки путем перебора всевозможных ключей, должно иметь строгую нижнюю оценку и выходить за пределы возможностей компьютеров,
4. знание алгоритма шифрования не должно влиять на надежность защиты
5. незначительное изменение ключа должно приводить к существенному изменению вида зашифрованного сообщения даже при использовании одного и того же ключа
6. структурные элементы алгоритма шифрования должны быть неизменными
7. дополнительные биты, вводимые в сообщение при шифровании должны быть полностью и надежно скрыты в шифрованном тексте
8. длина шифрованного текста должна быть равной длине исходного текста
9. не должно быть простых и легко устанавливаемых зависимостей между ключами, последовательно используемых при шифровании
10. любой ключ из множества возможных должен обеспечивать надежную защиту информации
11. алгоритм должен допускать как программную, так и аппаратную реализацию, при этом изменение длины ключа не должно вести к качественному ухудшению алгоритма шифрования.

Среди методов криптографического закрытия можно выделить следующие:
- Замена (подстановка)
- Перестановка
- Аналитическое преобразование
- Гаммирование
- Комбинированные методы

Гаммирование – наложение на текст псевдослучайной последовательности, генерируемой на основе ключа. Возможны следующие разновидности гамм:

- конечная короткая гамма,
- конечная длинная гамма,
- бесконечная гамма.

Шифрование методом гаммирования заключается в замене символов шифруемого текста и гаммы цифровыми эквивалентами (или в виде двоичного кода). Стойкость шифрования определяется длительностью периода и равномерностью статистических характеристик гаммы.

2. Выполнение работы

2.1. Задание на подготовку к работе

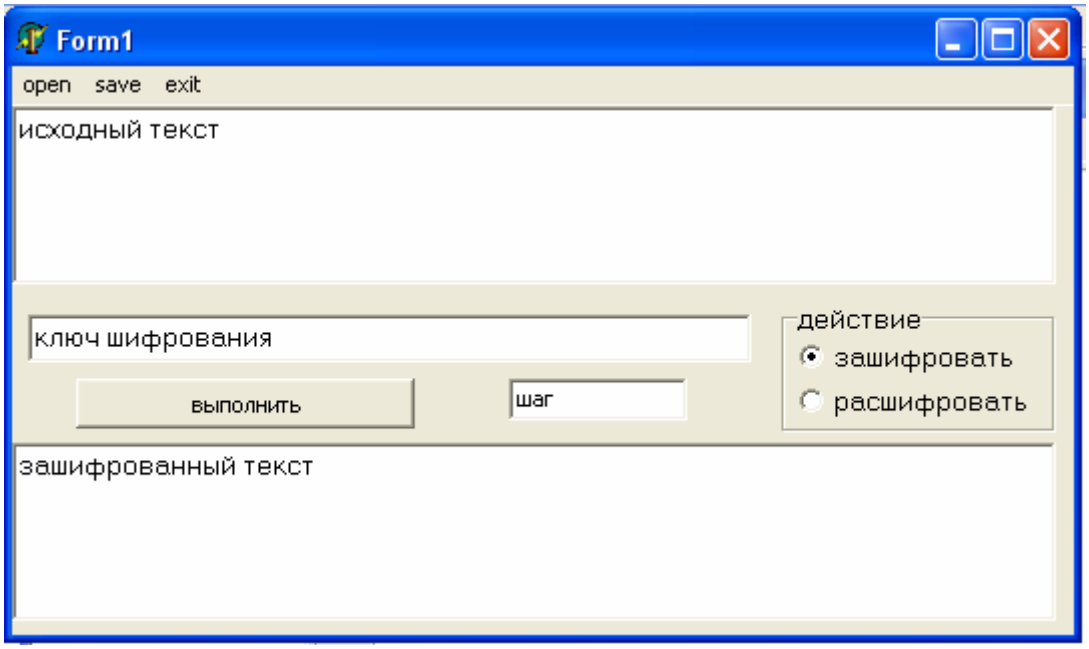
1. На своем профиле создать новую папку для хранения разрабатываемого приложения.
2. Запустить Delphi. Среда программирования создаст «пустой» проект.
3. Записать «пустой» проект в папку на своем профиле (пункт основного меню File->Save Project as)

2.2. Лабораторное задание

Задача лабораторной работы – разработать приложение, в котором вводимый текст шифруется симметричным алгоритмом с помощью задаваемого ключа.

3. Рекомендации при разработке приложения.

3.1 При разработке приложения используйте стандартные визуальные компоненты TMemo, TEdit, TButton, TRadioGroup, TMainMenu, TOpenDialog, TSaveDialog.
3.2 Рекомендуемый интерфейс приложения:



3.3 Возможный вариант процедур шифрования/дешифрования с оформлением каждого из действий в отдельную процедуру:

```
interface

const
  StartKey     = 981;          {Start default key}
  MultKey       = 12674;     {Mult default key}
  AddKey        = 35891;     {Add default key}

function Encrypt(const InString:string; StartKey,MultKey,AddKey:Integer): string;
function Decrypt(const InString:string; StartKey,MultKey,AddKey:Integer): string;

implementation

{$R-}
{$Q-}
{*******************************************************
 * Standard Encryption algorithm - Copied from Borland *
 *******************************************************}
function Encrypt(const InString:string; StartKey,MultKey,AddKey:Integer): string;
var
  I : Byte;
begin
  Result := '';
  for I := 1 to Length(InString) do
  begin
    Result := Result + CHAR(Byte(InString[I]) xor (StartKey shr 8));
    StartKey := (Byte(Result[I]) + StartKey) * MultKey + AddKey;
  end;
end;
{*******************************************************
 * Standard Decryption algorithm - Copied from Borland *
 *******************************************************}
function Decrypt(const InString:string; StartKey,MultKey,AddKey:Integer): string;
var
  I : Byte;
begin
  Result := '';
  for I := 1 to Length(InString) do
  begin
    Result := Result + CHAR(Byte(InString[I]) xor (StartKey shr 8));
    StartKey := (Byte(InString[I]) + StartKey) * MultKey + AddKey;
  end;
end;
{$R+}
{$Q+}

end.
```

3.4 Рекомендуется разработать процедуры шифрования/дешифрования с использованием наложения на шифруемый текст конечной короткой гаммы (ключа):
- символы текста и ключа переводятся в цифровой (тип Byte) аналог таблицы ASCII,
- организуется цикл перебора элементов массива текста с наложением на них

(сложение или вычитание с контролем выхода за границы типа данных) соответствующих элементов массива ключа.

3.5 Ключ шифрования перед записью в файл преобразуйте с помощью алгоритма шифрования XOR. Пример модуля консольного приложения с реализацией подобного алгоритма приведен ниже.

```
program Crypt;
{$APPTYPE CONSOLE}

uses Windows;

var
  key, text, longkey, result : string;
  i : integer;
  toto, c : char;
  F : TextFile;
begin
  writeln('Enter the key:');
  readln(key);
  writeln('Enter the text:');
  readln(text);

  for i := 0 to (length(text) div length(key)) do
    longkey := longkey + key;

  for i := 1 to length(text) do
  begin
    // XOR алгоритм
    toto := chr((ord(text[i]) xor ord(longkey[i])));
    result := result + toto;
  end;
  writeln('The crypted text is:');
  writeln(result);
  write('Should i save it to result.txt ?');
  read(c);
  if c in ['Y','y'] then
  begin
    AssignFile(F,'result.txt');
    Rewrite(F);
    Writeln(F,result);
    CloseFile(F);
  end;
end.
```

3.6. Разработайте приложение, в котором зашифрованный текст хранится в файле текстового формата вместе с ключом.

Перед записью в файл зашифрованного текста и ключа предварительно перемешайте записываемые массивы (в получаемом массиве выделяются фиксированные позиции для хранения символов ключа; для корректной сборки ключа необходимо указать длину ключа в символах и шаг фиксированных позиций, т.е. через сколько символов шифрованного текста находятся символы ключа; указанные признаки можно записать в качестве первых двух байтов файла или же в других позициях файла).

Расшифровка содержимого файла состоит из следующих шагов:

- считывание характеристик ключа,
- сборка ключа,
- расшифровка текста.

4. Содержание отчета

Отчет оформляется в форме Zip-архива, содержащего все файлы проекта (Project1.cfg, Project1.dof, Project1.dpr, Project1.exe, Project1.res, Unit1.dcu, Unit1.dfm, Unit1.pas). Zip-архив именуется по фамилии студента, номера группы и названия лабораторной работы.

5. Контрольные вопросы

1. Почему криптографические алгоритмы, требующие сохранения в тайне последовательности преобразования данных, не находят в настоящее время широкого применения?
2. Каким должен быть объем ключевого пространства для обеспечения криптографической стойкости алгоритма?
3. Зависит ли криптографическая стойкость алгоритма от набора возможных символов ключа?
4. Что такое ключ?
5. Можно ли использовать последовательности цифр фундаментальных констант при формировании гаммы?
6. Назовите основные показатели криптостойкости.
7. Охарактеризуйте меры по защите ключей.
8. Исходя из чего определяется необходимость смены ключей шифрования?
9. Назовите основную технико-экономическую характеристику систем криптографического закрытия.

.

Лабораторная работа N2

Защита программ от несанкционированной эксплуатации за счет привязки к носителю информации

Цель работы: получить навыки по установке защиты на разрабатываемые программы за счет привязки к носителю информации (устройства внешней памяти).

1. Основные положения.

Установление рыночных отношений в обществе вынуждает производителей защищать свой продукт от незаконного его использования. Особо актуальной эта проблема становится в области информационных технологий. Как показывает практика, абсолютных способов защиты информации не существует. Какими бы сложными и дорогими не были предлагаемые на рынке средства защиты, их эффективность оказывается условной. С учетом сложившейся реальной обстановки востребованными оказываются несложные и недорогие средства защиты, разрабатываемые и устанавливаемые самим производителем продукта и направленные против незаконных действий квалифицированных пользователей.

Идея защиты основывается на использовании индивидуальных характеристик носителей информации. При запуске приложения проводится проверка на наличие подключенного внешнего устройства с конкретным серийным номером и на нахождение стартовавшей программы на этом устройстве. Данный способ защиты позволяет беспрепятственно копировать приложение, существующие стандарты записи информации не нарушаются, специфических требований к устройствам считывания-записи нет. Возможна организация проверки из разных точек программы с использованием нескольких подобных процедур: цель данных действий – усложнить работу квалифицированного взломщика.

Среда ускоренной разработки приложений Delphi позволяет напрямую работать с функциями API-Windows.

Для получения информации об устройстве используется WinAPI функция **GetVolumeInformation**. В HELP-е Delphi (раздел Windows SDK) дано следующее описание параметров этой функции:

```
BOOL GetVolumeInformation(

  LPCTSTR lpRootPathName,              // address of root directory of the file system
  LPTSTR lpVolumeNameBuffer, // address of name of the volume
   DWORD nVolumeNameSize,              // length of lpVolumeNameBuffer
  LPDWORD lpVolumeSerialNumber,     // address of volume serial number
  LPDWORD lpMaximumComponentLength,      // address of system's maximum
                                  //filename length
 LPDWORD lpFileSystemFlags,  // address of file system flags
 LPTSTR lpFileSystemNameBuffer,       // address of name of file system
 DWORD nFileSystemNameSize            // length of lpFileSystemNameBuffer
);
```

Параметры функции:

lpRootPathName – имя устройства, информацию о котором необходимо получить;

lpVolumeNameBuffer – имя буфера, в который будет помещено имя тома;

nVolumeNameSize – размер буфера для имени тома;

lpVolumeSerialNumber – переменная, в которую будет записан серийный номер;

lpMaximumComponentLength – переменная, в которую будет записано максимальное значение пути, поддерживаемое файловой системой;

lpFileSystemFlags – флаги файловой системы, может быть любая комбинация флагов:

FS_CASE_IS_PRESERVED – файловая система сохраняет регистр имен файлов, когда сохраняет имя на диске;

FS_CASE_SENSITIVE – файловая система чувствительна к регистру имен файлов;

FS_UNICODE_STORED_ON_DISK – файловая система поддерживает имена в UNICODE;

FS_PERSISTENT_ACLS – файловая система поддерживает списки доступа (NTFS);

FS_FILE_COMPRESSION – файловая система поддерживает компрессию на уровне файлов;

FS_VOL_IS_COMPRESSED – файловая система поддерживает компрессию на уровне тома;

lpFileSystemNameBuffer – буфер, в который будет помещено имя файловой системы;

nFileSystemNameSize – размер буфера для имени файловой системы.

Типы рассмотренных параметров для корректного использования процедуры в среде Delphi:

```
lpRootPathName          :   PChar;
lpVolumeNameBuffer      :   PChar;
nVolumeNameSize         :   dWord;
lpVolumeSerialNumber    :   dWord;
lpMaximumComponentLength:   dWord;
lpFileSystemFlags       :   dWord;
lpFileSystemNameBuffer  :   PChar;
nFileSystemNameSize     :   dWord;
```

Начальная загрузка параметров:

```
lpVolumeNameBuffer      := '';
lpVolumeSerialNumber    := 0;
lpMaximumComponentLength:= 0;
lpFileSystemFlags       := 0;
lpFileSystemNameBuffer  := '';
GetMem(lpVolumeNameBuffer, Max_Path+1);     //выделение памяти для переменной
GetMem(lpFileSystemNameBuffer, Max_Path+1); //выделение памяти для переменной
nVolumeNameSize         := Max_Path+1;
nFileSystemNameSize     := Max_Path+1;
lpRootPathName  := PChar(DriveComboBox1.Drive+':\');
```

//имя устройства определяется компонентом TDriveComboBox

Вызов функции:

```
If GetVolumeInformation(
                        lpRootPathName,
                        lpVolumeNameBuffer,
                        nVolumeNameSize,
                         @lpVolumeSerialNumber,
                        lpMaximumComponentLength,
                        lpFileSystemFlags,
                         lpFileSystemNameBuffer,
                         nFileSystemNameSize)
 then begin …<действия> …end;
```

Рекомендации по проектированию защиты программ:

- не используйте стандартные обработчики компонентов, а организуйте проверки в цикле сообщений,
- не храните коды в одном месте,
- не проверяйте код только в одном месте,
- не анализируйте характеристику сразу после ее получения (считывания),
- не создавайте для проверки функцию или библиотеку,
- не задавайте действия, связанные с проверкой, сразу после самой проверки,
- применяйте отвлекающие функции проверок,
- не храните результаты проверок в переменных,
- не проверяйте контрольные данные одним алгоритмом,
- не храните результаты проверки в реестре,
- применяйте шифрование программ и данных,
- не записывайте текстовые строки в программе в их реальном виде.

Действия, направленные против использования отладчиков при взломе программ:

- определение отладчика до запуска программы с последующим завершением или эмуляцией ошибки,
- изменение работы программы в случае ее выполнения в отладчике,
- усложнение листинга,
- зашифрованные строки в ресурсах.

2. Выполнение работы

2.1. Задание на подготовку к работе

1. На своем профиле создать новую папку для хранения разрабатываемого приложения.
2. Запустить Delphi. Среда программирования создаст «пустой» проект.
3. Записать «пустой» проект в папку на своем профиле (пункт основного меню File->Save Project as)
4. Подключить к системному блоку внешний носитель информации (flash-память)

2.2. Лабораторное задание

Задача лабораторной работы –

а) определить серийный номер подключенного внешнего носителя информации,

б) встроить в приложение проверку наличия внешнего носителя с конкретным серийным номером.

3. Рекомендации при разработке приложения.

3.1. Для определения серийного номера внешнего устройства создайте отдельное приложение и используйте в нем стандартный компонент TDriveComboBox. Возможный вариант интерфейса вспомогательного приложения приведен на рис.1

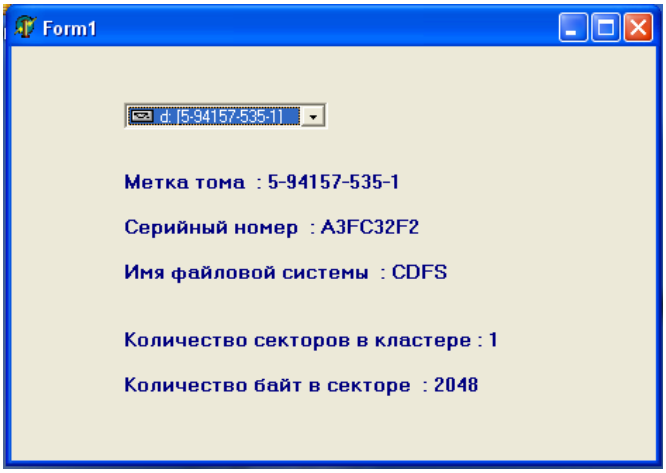


Рис.1 Внешний вид окна вспомогательного приложения

3.2. Для определения подключенных к компьютеру внешних устройств при встраивании защиты в программу используйте функции **GetLogicalDrives**: Integer и **GetDriveType**(Name: PChar): Integer.

Текст процедуры, выводящей в компонент TListBox список подключенных внешних устройств компьютера:

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
var i, mask: Integer;
    S: String;
begin
  mask := GetLogicalDrives;
  I := 0;
  while mask <> 0 do
  begin
    s := chr(ord('a') + i) + ':\';
    if (mask and 1) <>0 then
    case GetDriveType(PChar(S)) of
  0: ListBox1.Items.Add(S + 'unknow');
  1: ListBox1.Items.Add(S + 'not exists');
  Drive_Removable: ListBox1.Items.Add(S + 'removable');   //floppy
  Drive_Fixed: ListBox1.Items.Add(S + 'fixed');                  //hard
  Drive_Remote: ListBox1.Items.Add(S + 'network');
  Drive_CDROM: ListBox1.Items.Add(S + 'CD_ROM');          //cd
  Drive_RamDisk: ListBox1.Items.Add(S + 'RAM');
    end; //case
    inc(i);
    mask := mask shr 1;
  end;
end;
```

3.3 При проверке серийного номера внешнего устройства реализуйте вышеперечисленные рекомендации по проектированию защиты программ.

4. Содержание отчета

Отчет оформляется в форме Zip-архива, содержащего все файлы проекта (Project1.cfg, Project1.dof, Project1.dpr, Project1.exe, Project1.res, Unit1.dcu, Unit1.dfm, Unit1.pas). Zip-архив именуется по фамилии студента, номера группы и номера лабораторной работы.

5. Контрольные вопросы

1. Какие способы защиты от несанкционированного копирования программ Вы знаете?
2. Охарактеризуйте способ защиты, основанный на использовании меток носителя информации.
3. Охарактеризуйте способ защиты, основанный на физических дефектах носителя информации.
4. Охарактеризуйте способ защиты, основанный на временных характеристиках чтения носителя информации.
5. Как грамотно с точки зрения защиты от взлома представлять в исходном тексте программы шаблоны для сравнения строковых переменных?
6. Как организовать процедуры, выполняющие одни и те же действия, но имеющие разную «операторную начинку»?
7. Можно ли хранить значение серийного номера проверяемого устройства не в теле программы, а в отдельном файле?
8. Целесообразно ли выделять процедуры, осуществляющие действия по защите, в отдельные динамические библиотеки?
9. Как корректно именовать процедуры, осуществляющие защитные механизмы?
10. Как организовать шифрование строковых переменных в тексте программы?

Лабораторная работа N3

Программирование изменений характеристик файла

Цель работы: получить навыки по маскировке защитных действий в разрабатываемом приложении.

1. Основные положения.

Разработчик программы заинтересован в наиболее полном представлении возможностей своих разработок для привлечения внимания потенциального покупателя, в результате он вынужден рисковать, предлагая на рынок Trial и Demo-версии приложений, являющиеся полнофункциональными.

В качестве средств защиты от несанкционированной эксплуатации программного обеспечения широко используются способы распространения продукта с ограниченными возможностями, такими как
- ограниченный по времени период возможного использования продукта,
- ограниченное количество запусков программы.

При применении типовых методик в случае использования вышеупомянутых способов защиты, программисты ограничиваются записью контрольных параметров (счетчик запусков, предельная дата работы) непосредственно в реестр операционной системы. Для квалифицированного пользователя, работа в реестре не представляет больших затруднений. Как показала практика, упомянутый подход не является эффективным, так как место хранения контрольных параметров не является тайной.

Пример организации типовой защиты, основанной на ограничении количества запусков программы:

```
program Trial;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Registry, SysUtils, Dialogs; // Для TRegistry, IntToStr и ShowMessage

{$R *.RES}
var  N: Integer;
       Reg: TRegistry;
begin
  Reg := TRegistry.Create;
  with Reg do
  begin
    OpenKey('software', True);
    OpenKey('TrialProg', True);
    if ValueExists('MaxRun') then  // Первый запуск?
    begin              // - Нет
      N := ReadInteger('MaxRun')-1;
      if N>=0 then
        WriteInteger('MaxRun', N)
    end else begin     // -Да, первый запуск
      N := 5;
```

```
    WriteInteger('MaxRun', N)
   end;
   Free
  end;
  if N>0 then
  begin
   Application.CreateForm(TForm1, Form1);
   Form1.Label2.Caption := IntToStr(N-1);
   Application.Run;
  end else
   ShowMessage('Исчерпано максимальное количество запусков пробной версии
программы')
end.
```

При использовании защит, основанных на контроле даты выполнения или контроле количества запусков программы, более перспективным является хранение контрольных параметров в замаскированной форме в файлах, выбранных самим разработчиком и хранимых в произвольном месте (например, среди вспомогательных файлов самого приложения). При подобном подходе актуальной становится маскировка обращений к файлам, в которых хранятся текущие значения контрольных параметров. Общеизвестно, что при любом изменении содержимого файла операционная система автоматически корректирует внешние характеристики файла, такие, как длина, время и дата создания, которые хранятся в файловой системе.

Изменение даты и времени файла

Использование процедуры FileSetDate(Handle: Integer; newDate: TDateTime);

```
Handle – указатель (дескриптор) обрабатываемого файла, может быть определен с
помощью
   -  функции API-Windows FileOpen, например
        var Handle: Integer;
        . . . . . .
        Handle := FileOpen(filename, fmOpenReadWrite);
        FileSetDate(Handle, Datetimetofiledate(newtime));
        FileClose(Handle);

- свойства Handle класса TFileRec, например
        var TheFile: file;
        . . . . . .
        AssignFile(TheFile, FileName);
        Reset(TheFile);
        FileSetDate(TFileRec(TheFile).Handle, DateTimeToFileDate(newDate));
        Close(TheFile);
```

Использование процедуры SetFileTime

```
procedure ChangeDate(FileName: string);
var  i: TDate;
     Handle: Integer;
     f: TFileTime;
     s: TSystemTime;
begin
 Handle := CreateFile(PChar(FileName), $0100, 0, nil, OPEN_EXISTING,
  FILE_FLAG_BACKUP_SEMANTICS, 0);
```

```
  i := Date;

  DateTimeToSystemTime(i, S);
  SystemTimeToFileTime(S, F);
  LocalFileTimeToFileTime(F, F);
  SetFileTime(Handle, @f, @f, @f);
  CloseHandle(Handle);
end;
```

Использование процедуры SetFTime

```
var  f: file;
begin
  Assign(f, DirInfo.Name);
  Reset(f);
  SetFTime(f, Time);
  Close(f);
end;
```

2. Выполнение работы

2.1. Задание на подготовку к работе

1. На своем профиле создать новую папку для хранения разрабатываемого приложения.
2. Запустить Delphi. Среда программирования создаст «пустой» проект.
3. Записать «пустой» проект в папку на своем профиле (пункт основного меню File->Save Project as)

2.2. Лабораторное задание

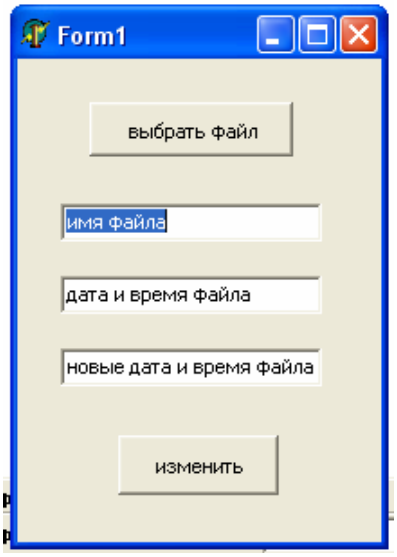
Задача лабораторной работы – организовать защиту, основанную на ограничении количества запусков приложения, причем значение счетчика запусков хранить в файле, внешние характеристики которого (дата и время создания) остаются постоянными.

3. Рекомендации при разработке приложения.

3.1 Разработайте тестовую программу, в которой в режиме диалога можно изменить дату создания выбранного файла

3.2 Для выбора файла используйте стандартные диалоговые компоненты TEdit, TButton, TOpenDialog.

3.3 Рекомендуемый интерфейс тестовой программы:



3.4 Возможный вариант организации процедуры для изменения даты и времени файла:

```
function SetFileDateTime(FileName: string; NewDateTime: TDateTime): Boolean;
 var
   Handle: Integer;
   FileTime: TFileTime;
   LFT: TFileTime;
   LST: TSystemTime;
 begin
   Result := False;
   try
     DecodeDate(NewDateTime, LST.wYear, LST.wMonth, LST.wDay);
     DecodeTime(NewDateTime, LST.wHour, LST.wMinute, LST.wSecond,
LST.wMilliSeconds);
     if SystemTimeToFileTime(LST, LFT) then
     begin
       if LocalFileTimeToFileTime(LFT, FileTime) then
       begin
         Handle := FileOpen(FileName, fmOpenReadWrite);
         if SetFileTime(Handle, nil, nil, @FileTime) then
           Result := True;
       end;
     end;
   finally
     FileClose(Handle);
   end;
 end;
```

3.5 Для маскировки действий используйте отвлекающие функции (мусорный код, например: организация «пустого» цикла с невыполняемыми переходами на лишние метки).

4. Содержание отчета

Отчет оформляется в форме Zip-архива, содержащего все файлы проекта (Project1.cfg, Project1.dof, Project1.dpr, Project1.exe, Project1.res, Unit1.dcu, Unit1.dfm, Unit1.pas). Zip-архив именуется по фамилии студента, номера группы и номера лабораторной работы.

5. Контрольные вопросы

1. Какие способы распространения программных продуктов Вы знаете?
2. В файле какого формата предпочтительнее хранить значение счетчика запусков программы?.
3. В каком виде предпочтительнее хранить значения даты и времени?
4. Как организовать в теле программы дополнительные проверки?
5. Что понимается под недокументированными точками входа в программу?
6. Сформулируйте рекомендации для обеспечения тестовых проверок работы приложения в случае использования защиты, основанной на использовании счетчика запусков программы.
7. Как рационально организовать ведение протокола при защите, основанной на учете количества запусков программы?
8. Как противостоять применению отладчиков при попытке взлома программы?
9. Как простыми средствами усложнить поиск точек останова программы?

Список рекомендуемой литературы

1. Партыка Т.Л., Попов И.И. Информационная безопасность. – М.: ФОРУМ: ИНФРА-М, 2004. – 368 с.
2. Скляров Д.В. Искусство защиты и взлома информации. – СПб.: БХВ-Петербург, 2004. – 288 с.
3. Фленов М.Е. Программирование в Delphi глазами хакера. СПб: БХВ-Петербург, 2004 – 368с.
4. Касперски К. Техника защиты компакт-дисков от копирования. СПб: БХВ-Петербург, 2004 – 464с.
5. Фаронов В.В. Delphi 6: учебный курс. СПб: Питер, 2002 – 512с.
6. Панов А.С. Реверсинг и защита программ от взлома. СПб: БХВ-Петербург, 2006 – 256с.