ВОЕННО-КОСМИЧЕСКАЯ АКАДЕМИЯ имени А.Ф.МОЖАЙСКОГО

Кафедра № 27 Математического и программного обеспечения

УТВЕРЖДАЮ

Начальник 27 кафедры

полковник

С. Войцеховский

«___» _____ 2022 г.

Практическое занятие № 3
по учебной дисциплине
«Защита информации»
на тему:

«**Защита программных средств от несанкционированного копирования, исследования, модификации**»

Рассмотрено и одобрено
на заседании кафедры № 27
« ___ » _____ 202_ г. протокол № ___

Санкт-Петербург
2022

I. ТЕМА И ЦЕЛЬ ПРАКТИЧЕСКОГО ЗАНЯТИЯ

Тема практического занятия: «Защита программных средств от несанкционированного копирования, исследования, модификации».

Учебная цель: овладение навыками составления и отладки модуля защиты ПО от модификации.

Время - 180 мин.

Место – аудитория (класс) по расписанию занятий.

Учебно-материальное и методическое обеспечение

1. Лабораторные установки – персональные ЭВМ с установленным на них программным обеспечением.
2. Методические разработки по программированию модулей защиты ПО от модификации.
3. Варианты типовых заданий на практическое занятие.

II. УЧЕБНЫЕ ВОПРОСЫ И РАСЧЕТ ВРЕМЕНИ

№ п\п	Учебные вопросы	Время, мин.
1.	Вступительная часть. Контрольный опрос.	10
2.	Учебные вопросы. ОСНОВНАЯ ЧАСТЬ: 1. Разработка программного модуля для защиты ПО от модификации. 2. Проверка работоспособности модуля путём проверки требуемого файла. 3. Составление отчёта о проделанной работе, защита программы у преподавателя.	80 40 45
3.	Заключительная часть. Задание и методические указания курсантам на самостоятельную подготовку	5

III. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПРЕПОДАВАТЕЛЮ ПРИ ПРОВЕДЕНИИ ПРАКТИЧЕСКОГО ЗАНЯТИЯ

Во вступительной части занятия производится контроль присутствия и готовности обучающихся к занятию. Объявляется тема, цель, учебные вопросы занятия и особенности его проведения.

Готовность группы к занятию проверяется контрольным опросом.

Вопрос 1: Что подразумевают под понятием защита от модификации?

Вопрос 2: Какими способами организуются защита от модификации?

Вопрос 3: Для чего необходим механизм защиты от модификации?

При отработке первого вопроса занятия основное внимание обратить на усвоение обучающимися принципов построения модулей защиты программ от модификации.

При отработке второго вопроса отметить необходимость и важность встраивания модуля защиты от модификации в структуру любой программы, как решающего фактора своевременного и правильного решения задачи защиты ПО.

При отработке третьего вопроса необходимо акцентировать внимание на структуре отчета о проделанной работе и защите его основных положений.

В заключительной части занятия подвести итоги, оценить действия обучающихся, ответить на вопросы.

Дать задание на самоподготовку. Объявить тему следующего занятия.

IV. УЧЕБНЫЕ МАТЕРИАЛЫ

1. Сведения из теории

Контрольная сумма – некоторое значение, рассчитанное по набору данных путём применения определённого алгоритма и используемое для проверки целостности данных при их передаче или хранении. Также контрольные суммы могут использоваться для быстрого сравнения двух наборов данных на неэквивалентность: с большой вероятностью различные наборы данных будут иметь неравные контрольные суммы. Это может быть использовано, например, для обнаружения компьютерных вирусов. Несмотря на своё название, контрольная сумма не обязательно вычисляется путем суммирования.

С точки зрения математики контрольная сумма является результатом хеш-функции, используемой для вычисления контрольного кода – небольшого количества бит внутри большого блока данных, например, сетевого пакета или блока компьютерного файла, применяемого для обнаружения ошибок при передаче или хранении информации. Значение контрольной суммы добавляется в конец блока данных непосредственно перед началом передачи или записи данных на какой-либо носитель информации. Впоследствии оно проверяется для подтверждения целостности данных.

Популярность использования контрольных сумм для проверки целостности данных обусловлена тем, что подобная проверка просто реализуема в двоичном цифровом оборудовании, легко анализируется и хорошо подходит для обнаружения общих ошибок, вызванных наличием шума в каналах передачи данных.

Криптографическая функция MD5 уже почти не используется для определения контрольных сумм, т.к. оказалось, что для неё можно быстро создавать с помощью современных компьютеров два разных файла, имеющих разную длину в байтах, но одинаковые величины контрольных сумм, подсчитанных с помощью алгоритма MD5.

Использование термина сумма связано с тем, что на заре цифровой связи при байтовых передачах информационными были 7 бит, а восьмой – контрольный – рассчитывался как младший разряд сложения информационных.

Примеры

Циклический избыточный код (в частности, CRC8, CRC16, CRC32) применяется для проверки целостности передачи данных. Программы-архиваторы включают CRC исходных данных в созданный архив для того, чтобы получающий мог удостовериться в корректности полученных данных. Такая контрольная сумма проста в реализации и обеспечивает низкую вероятность возникновения коллизий.

MD5 и другие криптографические хеш-функции используются, например, для подтверждения целостности и подлинности передаваемых данных.

Под названием «контрольное число» входит в состав номеров товаров и различных документов.

2. Пример разработки программного модуля

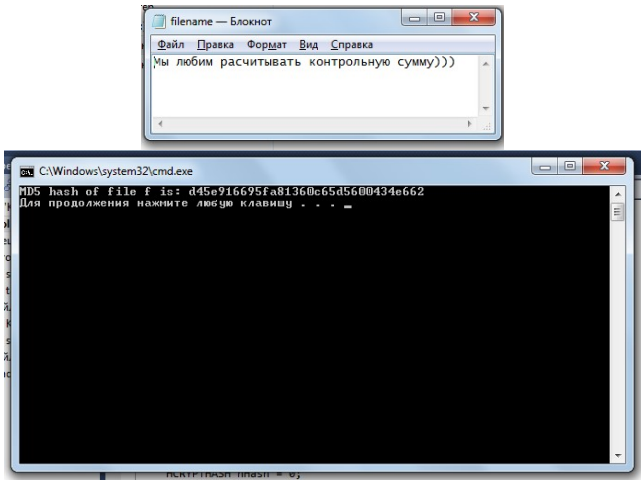
Kontrol sum.cpp: пример демонстрирует использование CryptoAPI которая вычисляет MD5-хеша содержания файла. Этот пример выполняет вычисление на содержимое файла, указанного во время выполнения.

```
//

#include "stdafx.h"
#include <stdio.h>
#include <windows.h>
#include <Wincrypt.h>

#define BUFSIZE 1024
#define MD5LEN  16

DWORD main()
{
   DWORD dwStatus = 0;
   BOOL bResult = FALSE;
   HCRYPTPROV hProv = 0;
   HCRYPTHASH hHash = 0;
   HANDLE hFile = NULL;
   BYTE rgbFile[BUFSIZE];
   DWORD cbRead = 0;
   BYTE rgbHash[MD5LEN];
   DWORD cbHash = 0;
   CHAR rgbDigits[] = "0123456789abcdef";
   LPCWSTR filename=L"filename.txt";
   // Logic to check usage goes here.

   hFile = CreateFile(filename,
      GENERIC_READ,
      FILE_SHARE_READ,
      NULL,
      OPEN_EXISTING,
      FILE_FLAG_SEQUENTIAL_SCAN,
      NULL);

   if (INVALID_HANDLE_VALUE == hFile)
   {
      dwStatus = GetLastError();
      printf("Error opening file %s\nError: %d\n", filename,
         dwStatus);
      return dwStatus;
   }
```

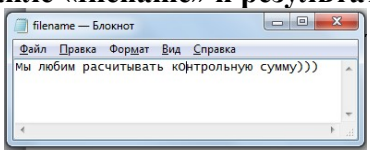
```
// Get handle to the crypto provider
if (!CryptAcquireContext(&hProv,
   NULL,
   NULL,
   PROV_RSA_FULL,
   CRYPT_VERIFYCONTEXT))
{
   dwStatus = GetLastError();
   printf("CryptAcquireContext failed: %d\n", dwStatus);
   CloseHandle(hFile);
   return dwStatus;
}

if (!CryptCreateHash(hProv, CALG_MD5, 0, 0, &hHash))
{
   dwStatus = GetLastError();
   printf("CryptAcquireContext failed: %d\n", dwStatus);
   CloseHandle(hFile);
   CryptReleaseContext(hProv, 0);
   return dwStatus;
}

while (bResult = ReadFile(hFile, rgbFile, BUFSIZE,
   &cbRead, NULL))
{
   if (0 == cbRead)
   {
      break;
   }

   if (!CryptHashData(hHash, rgbFile, cbRead, 0))
   {
      dwStatus = GetLastError();
      printf("CryptHashData failed: %d\n", dwStatus);
      CryptReleaseContext(hProv, 0);
      CryptDestroyHash(hHash);
      CloseHandle(hFile);
      return dwStatus;
   }
}

if (!bResult)
{
   dwStatus = GetLastError();
   printf("ReadFile failed: %d\n", dwStatus);
   CryptReleaseContext(hProv, 0);
   CryptDestroyHash(hHash);
   CloseHandle(hFile);
```

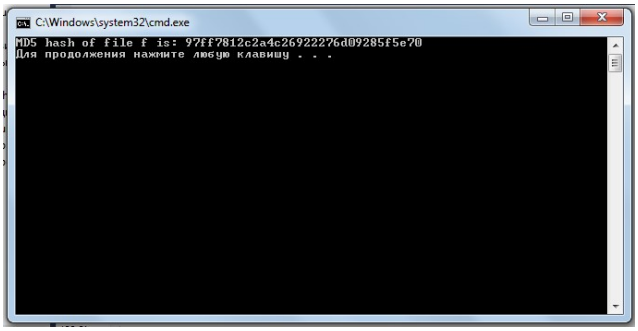
```
    return dwStatus;
}

cbHash = MD5LEN;
if (CryptGetHashParam(hHash, HP_HASHVAL, rgbHash, &cbHash, 0))
{
    printf("MD5 hash of file %s is: ", filename);
    for (DWORD i = 0; i < cbHash; i++)
    {
        printf("%c%c", rgbDigits[rgbHash[i] >> 4],
            rgbDigits[rgbHash[i] & 0xf]);
    }
    printf("\n");
}
else
{
    dwStatus = GetLastError();
    printf("CryptGetHashParam failed: %d\n", dwStatus);
}

CryptDestroyHash(hHash);
CryptReleaseContext(hProv, 0);
CloseHandle(hFile);
return dwStatus;
      }
```

Исходный текст и результат расчета контрольной суммы файла «filename»!



Измененный текст в том же файле «filename» и результат расчета контрольной суммы!





3.Алгоритм контрольного суммирования CRC

Алгоритм контрольного суммирования CRC расшифровывается, как циклический избыточный код *(Cyclic redundancy code)*, и предназначается для контроля целостности данных. Он широко используется в проводных и беспроводных сетях, и в устройствах хранения данных, для проверки информации на подлинность и защиты от несанкционированного изменения. Он основывается на свойствах деления с остатком многочлена на многочлен. По сути, результатом контрольного суммирования CRC является остаток от деления многочлена, соответствующего исходным данным, на порождающий многочлен фиксированной длины.

Очевидно, что количество различных остатков от деления многочлена на многочлен меньше, чем количество различных исходных многочленов. Таким образом, контрольное суммирование CRC может однозначно дать ответ, что два массива данных отличаются друг от друга, если отличаются их контрольные суммы. Но, если две контрольные суммы совпали, нельзя однозначно утверждать, что для их формирования использовался один и тот же исходный массив данных.

В зависимости от вида порождающего многочлена и его длины, изменяется вероятность совпадения контрольных сумм для различных исходных данных и время контрольного суммирования. Наиболее популярными являются алгоритмы CRC, работающие с порождающими многочленами: восьмой (CRC-8), шестнадцатой (CRC - 16) и тридцать второй (CRC – 32) степени.

Выбор длины порождающего многочлена, кратной байту, позволяет ускорить работу программы по контрольному суммированию, обеспечивая достаточную надежность полученного результата. Например, контрольное суммирование CRC-32 в пределе позволяет получить надежность порядка: 2^{32}= 4.294.967.296. Что в принципе позволяет, практически со 100% вероятностью, обнаруживать сбои при хранении и передаче данных.

Существует достаточно большое разнообразие порождающих многочленов для алгоритмов контрольного суммирования CRC – 8, 16 и 32, подобранных на основе теории кодирования и многочисленных исследований. Ниже приведены некоторые из них:

CRC-8: $x^8 + x^7 + x^6 + x^4 + x^2 + 1$ –и Используется формой Dallas Semiconductor в устройствах низкоскоростной связи.

CRC-16: $x^{16} + x^{15} + x^2 + 1$ - используется в таких интерфейсах, как USB, ModBus и других линиях связи.

CRC-32: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ - используется при кодировании видео и аудио сигналов с использованием стандарта MPEG-2, при кодировании растровых изображений в формате PNG и во многих других случаях.

В вычислительной технике оперировать с полиномами N–степени - неудобно и ресурсоемко. Поэтому полиномы заменяют бинарными последовательностями и вычисляют контрольную сумму, оперируя уже не с полиномами, а с бинарными данными.

Действительно, любому полиному можно однозначно сопоставить бинарную последовательность. Если полином в общем виде записывается, как $A_1*X^n + A_2*X^{n-1} +\ldots+ A_{n-1}*X + A_n$, где коэффициенты A_1 ... A_n принимают значения единицы или нуля, то достаточно записать последо-

вательность из коэффициентов A_1 ... A_n, чтобы однозначно задать полином. Например, полином $X^4 + X^2 + 1$ однозначно соответствует бинарной последовательности 10101, так как $1*X^4 + 0*X^3 + 1*X^2 + 0*X + 1 = X^4 + X^2 + 1$.

Так как деление можно заменить повторением операций вычитания, рассмотрим, как осуществляется вычитание в полиномиальной арифметике по модулю 2.

Полиномиальная арифметика по модулю 2 — это один из видов арифметики, используемый для решения задач в определенной предметной области и отличающийся от привычной, двоичной арифметики с циклическим переносом, отсутствием переносов и вычислением всех коэффициентов по модулю 2.

Таким образом, вычитание полиномов сводится к операции «исключающего или» с элементами полинома, имеющими одну и ту же степень, а, следовательно, мы можем заменить вычитание полиномов на операцию «исключающие или», с сопоставленными им бинарными последовательностями. Рассмотрим это утверждение на примере вычитания из полинома $X^4 + X^2+1$ полинома $X^3 + X^2$ (операцию "исключающее или" обзначим значком ‘^’, как это принято в языке Си):

$$\frac{X^4+X^2+1}{X^3+X^2} = X^4+X^3+1 \equiv \frac{10101}{1100} = 11001$$

Вычитание полиномов

Используя описанную выше возможность замены полинома на бинарную последовательность, рассмотрим алгоритм подсчета контрольной суммы CRC8:

Исходный массив данных: 1001 0110 0100 1011.

Порождающий многочлен: 1101 0101.

Пример расчета контрольной суммы CRC - 8

Аналогичным способом подсчитываются контрольные суммы CRC-16, CRC-32, CRC-64 и т.д. Как видите, алгоритмы очень просты и легко реализуются на любой ЭВМ.

Особенно важно здесь, что работаем мы не со всеми данными, а только с небольшой последовательностью битов (для CRC-8 – с 8-ю битами, для CRC-16 – с 16-ю битами), затем, сдвигаясь на один бит, опять работаем с небольшой последовательностью битов такой же длины. Это позволяет нам легко обрабатывать огромные массивы данных, не загружая их полностью в память и не расходуя понапрасну вычислительные ресурсы.

Обладая простой реализацией и, в то же время, обеспечивая высокую надежность, алгоритм контрольного суммирования CRC завоевал большую популярность. Существует огромное количество разнообразных программ, использующих этот алгоритм и различными способами оптимизирующих скорость подсчета контрольной суммы.

4. Вычисление контрольной суммы файла

В примере демонстрируется использование библиотеки классов java.util.zip для вычиления контрольной суммы файла по методам CRC-32 и Adler-32.

Иногда перед программистом встает задача вычисления контрольной суммы файла. Такая сумма может потребоваться, например, чтобы убедиться в сохранности файла при его передаче по сети или при его хранении в небезопасных условиях.

Разумеется, вы можете сами организовать подсчет контрольной суммы, например, просто складывая последовательности четырехбайтовых слов по модулю два или с применением какого-либо иного алгоритма.

В библиотеке классов java.util.zip есть классы и интерфейсы, намного упрощающие данную задачу. Это классы CRC32, Adler32 и интерфейс Checksum.

Интерфейс Checksum

Интерфейс Checksum определяет методы, применяющиеся при подсчете контрольной суммы. Это методы update, getValue и reset.

Метод update существует в виде двух реализаций:

```
public abstract void update(int b);
public abstract void update(byte b[],
  int off, int len);
```

Первая из них обновляет контрольную сумму значением одного байта, а вторая - значениями массива байт.

Метод getValue возвращает текущее значение контрольной суммы:

```
public abstract long getValue();
```

Метод reset сбрасывает значение контрольной суммы в исходное состояние:

```
public abstract void reset();
```

5. Классы CRC32 и Adler32

Классы CRC32 и Adler32 предоставляют в ваше распоряжение две различные готовые реализации интерфейса Checksum, использующие 32-разрядное суммирование. Первый из них реализует алгоритм CRC-32, а второй - Adler-32 (более быстродействующий).

Для того чтобы организовать в своем приложении вычисление контрольной суммы файла с применением классов CRC32 или Adler32, вам вначале нужно создать объект одного из этих классов, воспользовавшись для этого соответствующим конструктором. Заметим, что конструкторы этих классов не имеют параметров.

Далее вам нужно создать входной поток для исследуемого файла и организовать его чтение блоками в цикле. На каждой итерации цикла вы должны вызывать метод update класса CRC32 или Adler32.

После завершения цикла искомая контрольная сумма может быть получена методом getValue, определенным в этих классах.

Описание примера

Наше приложение позволяет выбирать файлы с помощью стандартной диалоговой панели класса FileDialog, определяя для них контрольные суммы с применением алгоритмов CRC-32 и Adler-32.

Путь к файлу и значения контрольных сумм отображаются в окне многострочного редактора текста, расположенного в главном окне приложения (рис. 1).

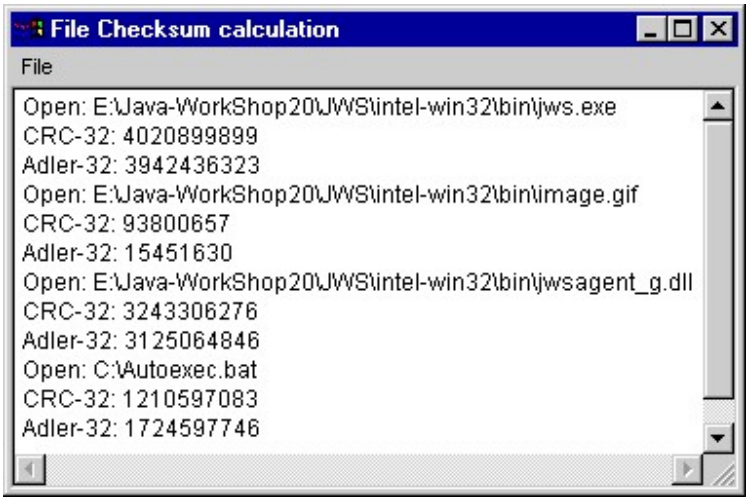


Рис. 1. Просмотр контрольных сумм файлов

Рассмотрим наиболее важные фрагменты исходного текста приложения.

Когда пользователь выбирает из меню File строку Open, управление передается методу actionPerformed.

Данный метод создает и отображает на экране диалоговую панель выбора входного файла, получает путь к выбранному пользователем файлу, сохраняет его в переменной szPath и показывает в окне приложения:

```
if(e.getSource().equals(miOpen))
{
 fdlg = new FileDialog(this, "Open file",
  FileDialog.LOAD);
 fdlg.show();

 String szPath = fdlg.getDirectory() +
  fdlg.getFile();

 ta.append("Open: " + szPath  + "\n");
 . . .
}
```

Далее мы два раза вызываем метод calculate, определенный в нашем приложении, который возвращает текстовое представление значения контрольной суммы:

```
ta.append("CRC-32: " +
 calculate(szPath, new CRC32()) + "\n");

ta.append("Adler-32: " +
 calculate(szPath, new Adler32()) + "\n");
```

Через первый параметр этому методу передается путь к файлу, а через второй - ссылка на класс, определяющий метод вычисления контрольной суммы.

Как устроен метод calculate?

Прежде всего этот метод открывает входной поток для файла, контрольную сумму которого необходимо вычислить:

```
String s = "";
byte[] buf = new byte[8000];
int nLength = 0;
```

```
try
{
 FileInputStream fis =
   new FileInputStream(szPath);
 . . .
}
```

Далее мы организуем чтение из потока в цикле блоками по 8000 байт (значение, выбранное нами произвольно), с периодическим обновлением контрольной суммы методом update:

```
while(true)
{
 nLength = fis.read(buf);
 if(nLength < 0)
   break;

 cs.update(buf, 0, nLength);
}
```

После достижения конца файла связанный с ним поток закрывается, а значение контрольной суммы извлекается методом getValue:

```
fis.close();
. . .
s = new Long(cs.getValue()).toString();
return s;
```

Мы преобразуем его в текстовую строку класса String и возвращаем вызвавшему методу.

6.Исходный текст программы GetChecksum.java

```
// ========================================
// GetChecksum.java
// ========================================
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.zip.*;

public class GetChecksum
{
 public static void main(String args[])
 {
  FrameWindow frame;
  frame =  new FrameWindow(
    "File Checksum calculation");
  frame.setVisible(true);
 }
}

// ========================================
```

```
// Class FrameWindow
// ======================================
class FrameWindow extends Frame
 implements ActionListener, WindowListener
{
 TextArea ta;

 MenuBar mb;

 Menu mFile;
 MenuItem miOpen;
 MenuItem miExit;

 FileDialog fdlg;

 // ============================================
 // FrameWindow
 // ============================================
 public FrameWindow(String szTitle)
 {
  super(szTitle);
  setSize(400, 300);

  mb = new MenuBar();
  mFile = new Menu("File");

  miOpen = new MenuItem("Open...");
  mFile.add(miOpen);

  mFile.add("-");

  miExit = new MenuItem("Exit");
  mFile.add(miExit);

  mb.add(mFile);

  miOpen.addActionListener(this);
  miExit.addActionListener(this);

  setMenuBar(mb);

  this.addWindowListener(this);

  ta = new TextArea(10, 30);

  setLayout(new BorderLayout());
  add("Center", ta);
 }
```

```
// ============================================
// actionPerformed
// ============================================
public void actionPerformed(ActionEvent e)
{
 if(e.getSource().equals(miOpen))
 {
  fdlg = new FileDialog(this, "Open file",
         FileDialog.LOAD);
  fdlg.show();

  String szPath = fdlg.getDirectory() +
   fdlg.getFile();

  ta.append("Open: " + szPath  + "\n");

  ta.append("CRC-32: " +
         calculate(szPath, new CRC32()) + "\n");

  ta.append("Adler-32: " +
         calculate(szPath, new Adler32()) + "\n");
 }

 else if(e.getSource().equals(miExit))
 {
  setVisible(false);
  System.exit(0);
 }
}

// ============================================
// windowClosing
// ============================================
public void windowClosing(WindowEvent e)
{
 setVisible(false);
 System.exit(0);
}

public void windowOpened(WindowEvent e) {}
public void windowClosed(WindowEvent e) {}
public void windowIconified(WindowEvent e) {}
public void windowDeiconified(WindowEvent e) {}
public void windowActivated(WindowEvent e) {}
public void windowDeactivated(WindowEvent e) {}

// ============================================
```

```
 // calculate
 // ========================================
 String calculate(String szPath, Checksum cs)
 {
  String s = "";
  byte[] buf = new byte[8000];
  int nLength = 0;

  try
  {
   FileInputStream fis =
    new FileInputStream(szPath);

   while(true)
   {
          nLength = fis.read(buf);
          if(nLength < 0)
           break;

    cs.update(buf, 0, nLength);
   }

   fis.close();
  }
  catch(Exception ex)
  {
   System.out.println(ex.toString());
  }

  s = new Long(cs.getValue()).toString();
  return s;
 }
}
```

7.Общие методические указания курсантам (слушателям) по подготовке к практическим занятиям

Практические занятия по дисциплине «Защита информации» проводятся в классе ПЭВМ. Индивидуальные задания выполняются каждым курсантом лично.

Перед выполнением задания обучающийся изучает материал, приведенный в разделе «Учебные материалы», в ходе которого необходимо разобрать приведенные примеры и выполнить задания раздела. На следующем этапе работы обучающийся выполняет индивидуальное задание.

Результаты работы оформляются в виде отчета. Содержание отчета приведено в руководстве по соответствующему практическому занятию.

По готовности к защите работы курсант (слушатель) докладывает преподавателю.

8. Индивидуальные задания к практическому занятию

1. Разработать программный модуль реализующий расчет контрольной суммы файла *.dat с применением алгоритмов CRC* согласно нижеприведенной таблице:

№ по журналу	Алгоритм нахождения контрольной суммы (CRC)	Полином	Представления: нормальное / реверсированное / реверсированное от обратного
1.	CRC-4-ITU	$x^4 + x + 1$ (ITU G.704)	0x3 / 0xC / 0x9
2.	CRC-5-EPC	$x^5 + x^3 + 1$ (Gen 2 RFID)	0x09 / 0x12 / 0x14
3.	CRC-5-ITU	$x^5 + x^4 + x^2 + 1$ (ITU G.704)	0x15 / 0x15 / 0x1A
4.	CRC-5-USB	$x^5 + x^2 + 1$ (USB token packets)	0x05 / 0x14 / 0x12
5.	CRC-6-ITU	$x^6 + x + 1$ (ITU G.704)	0x03 / 0x30 / 0x21
6.	CRC-7	$x^7 + x^3 + 1$ (системы телекоммуникации, ITU-T G.707, ITU-T G.832, MMC, SD)	0x09 / 0x48 / 0x44
7.	CRC-8-CCITT	$x^8 + x^2 + x + 1$ (ATM HEC), ISDN Header Error Control and Cell Delineation ITU-T I.432.1 (02/99)	0x07 / 0xE0 / 0x83
8.	CRC-8-Dallas/Maxim	$x^8 + x^5 + x^4 + 1$ (1-Wire bus)	0x31 / 0x8C / 0x98
9.	CRC-8	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$ (ETSI EN 302 307, 5.1.4)	0xD5 / 0xAB /0xEA
10.	CRC-8-SAE J1850	$x^8 + x^4 + x^3 + x^2 + 1$	0x1D / 0xB8 / 0x8E
11.	CRC-10	$x^{10} + x^9 + x^5 + x^4 + x + 1$	0x233/ 0x331/0x319
12.	CRC-11	$x^{11} + x^9 + x^8 + x^7 + x^2 + 1$ (FlexRay)	0x385/0x50E/0x5C2
13.	CRC-12	$x^{12} \mid x^{11} \mid x^3 \mid x^2 \mid x \mid 1$ (системы телекоммуникации)	0x80F / 0xF01 / 0xC07
14.	CRC-15-CAN	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$	0x4599 / 0x4CD1 / 0x62CC
15.	CRC-16-IBM	$x^{16} + x^{15} + x^2 + 1$ (Bisync, Modbus, USB, ANSI X3.28, многие другие; также известен как CRC-16 и CRC-16-ANSI)	0x8005/0xA001/ 0xC002
16.	CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$ (X.25, HDLC, XMODEM, Bluetooth, SD и др.)	0x1021/0x8408 / 0x8810
17.	CRC-16-T10-	$x^{16} + x^{15} + x^{11} + x^9 + x^8 + x^7 - x^5 + x^4 + x^2 + x + 1$ (SCSI DIF)	0x8BB7/0xEDD1 / 0xC5DB

	DIF		
18.	CRC-16-DNP	$x^{16}+x^{13}-x^{12}+x^{11}+x^{10}+x^{8}+x^{6}+x^{5}+x^{2}+1$ (DNP, IEC 870, M-Bus)	0x3D65 / 0xA6BC / 0x9EB2
19.	CRC-24	$x^{24}+x^{22}+x^{20}+x^{19}+x^{18}+x^{10}+x^{14}+x^{13}+x^{11}+x^{10}+x^{8}+x^{7}+x^{6}+x^{3}+x+1$(FlexRay)	0x5D6DCB / 0xD3B6BA / 0xAEB6E5
20.	CRC-24-Radix-64	$x^{24}+x^{23}+x^{18}+x^{17}+x^{14}+x^{11}+x^{10}+x^{7}+x^{6}+x^{5}+x^{4}+x^{3}+x+1$ (OpenPGP)	0x864CFB/ 0xDF3261/ 0xC3267D

2. Проверка работоспособности модуля путём проверки контрольной суммы любого выбранного файла.

3. Подготовить отчет по проделанной работе.

9.Отчетность по работе

По выполнению работы каждый курсант должен представить отчет. Отчет должен содержать:

– название практического занятия;

– текст индивидуального задания;

– блок-схему алгоритма решения задачи;

– исходный текст программы;

– результаты тестирования решения.

В процессе выполнения индивидуального задания или после завершения его выполнения преподаватель проводит собеседование с каждым курсантом по теме выполненной работы, проверяя также практические навыки, приобретенные в ходе занятия. Отчетный материал предоставляется преподавателю, а результаты защищаются.

10. Заключительная часть

В заключительной части подводятся итоги проделанной работы, дается краткая оценка действиям участников, прослеживается связь с теоретическими положениями и перспективой на будущую деятельность.

11. Задание и методические указания курсантам на самостоятельную подготовку:

1. Повторить по конспекту лекций и рекомендованной литературе основные методы защиты от модификации.
2. Быть готовыми к самостоятельному составлению программ с использованием программных модулей защиты от модификации.

V. ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

1. Информационная безопасность: – учебное пособие / В.М.Зима, СПб.: ВКА имени А.Ф.Можайского, 2017 с.
2. Принципы построения и функционирования аппаратно-программных средств телекоммуникационных систем. Ч.2. Сетевые ОС и принципы обеспечения информационной безопасности в сетях / С.И. Макаренко, А.А. Ковальский, С.А. Краснов СПб.: Наукоемкие технологии 2020.

Доцент 27 кафедры
к.т.н.
подполковник

С. Краснов

«___»_____20__г.