Методические указания
к лабораторной работе №1

«Стеганография»

1. Цель работы:

Изучение принципов сокрытия следов присутствия информации.

2 Задачи работы:

Подготовка открытого текста к сокрытию.

Выбор типа цифрового контейнера и анализ его структуры.

Сокрытие текста в контейнер.

Разработка мер по повышению надежности сокрытия с следов присутствия информации.

Извлечение текста из контейнера.

3. Постановка задачи:

Разработать программное средство, автоматизирующее процесс сокрытия текста в цифровой контейнер по методу LSB.

ПС должно выполнять следующие функции:

- представление выбранного текста в двоичном виде;
- побитовое сохранение исходного текста в наименее значимый бит блоков цифрового контейнера;
- извлечение текста из контейнера.

4 Требования к отчету:

Отчет о проделанной работе должен содержать:

- название работы
- постановка задачи;
- описание структуры bmp-файла;
-обоснование и выбор метода хранения длины открытого текста в файле контейнере;
-обоснование и выбор метода повышения надежности метода LSB.
-руководство пользователя.

Отчет представить в электронном виде.

5 Общие положения и методика выполнения задания

LSB (Least Significant Bit, наименьший значащий бит) — суть этого метода заключается в замене последних значащих битов в контейнере (изображения, аудио или видеозаписи) на биты скрываемого сообщения. Разница между пустым и заполненным контейнерами должна быть не ощутима для органов восприятия человека.

Суть метода заключается в следующем: Допустим, имеется 8-битное изображение в градациях серого. 00h (00000000b) обозначает чёрный цвет, FFh (11111111b) — белый. Всего имеется 256 градаций (2^8). Также предположим, что сообщение состоит из 1 байта — например, 01101011b. При использовании 2 младших бит в описаниях пикселей, нам потребуется 4 пикселя. Допустим, они чёрного цвета. Тогда пиксели, содержащие скрытое сообщение, будут выглядеть следующим образом: 00000<u>01</u> 00000<u>10</u> 00000<u>10</u> 00000<u>11</u>. Тогда цвет пикселей изменится: первого — на 1/255, второго и третьего — на 2/255 и четвёртого — на 3/255. Такие градации, мало того что незаметны для человека, могут вообще не отобразиться при использовании низкокачественных устройств вывода.

Методы LSB являются неустойчивыми ко всем видам атак и могут быть использованы только при отсутствии шума в канале передачи данных.

Обнаружение LSB-кодированного стего осуществляется по аномальным характеристикам распределения значений диапазона младших битов отсчётов цифрового сигнала.

Все методы LSB являются, как правило, аддитивными (A17, L18D).

Другие методы скрытия информации в графических файлах ориентированы на форматы файлов с потерей, к примеру, JPEG. В отличие от LSB они более устойчивы к геометрическим преобразованиям. Это получается за счёт варьирования в широком диапазоне качества изображения, что приводит к невозможности определения источника изображения.

BMP (от англ. *Bitmap Picture*) — формат хранения растровых изображений, разработанный компанией Microsoft.

С форматом BMP работает огромное количество программ, так как его поддержка интегрирована в операционные системы Windows и OS/2. Файлы формата BMP могут иметь расширения `.bmp`, `.dib` и `.rle`. Кроме того, данные этого формата включаются в двоичные файлы ресурсов RES и в PE-файлы.

Компания Microsoft так же разработала для своих нужд форматы ICO и CUR, которые имеют похожую на BMP структуру. Кроме этого, структуры из этого формата используются некоторыми WinAPI-функциями подсистемы GDI.

Глубина цвета в данном формате может быть 1, 2, 4, 8, 16, 24, 32, 48 бит на пиксель, но глубина 2 бита на пиксель официально не поддерживается. При этом для глубины цвета меньше 16 бит используется палитра с полноцветными компонентами глубиной 24 бита.

В формате BMP изображения могут храниться как есть или же с применением некоторых распространённых алгоритмов сжатия. В частности, формат BMP поддерживает RLE-сжатие без потери качества, а современные операционные

системы и программное обеспечение позволяют использовать JPEG и PNG (эти форматы встраиваются в BMP как в контейнер).

BMP-файл состоит из четырёх частей:

1. Заголовок файла (BITMAPFILEHEADER)
2. Заголовок изображения (BITMAPINFOHEADER, может отсутствовать). BITMAPV4HEADER (Win95, NT4.0) BITMAPV5HEADER (Win98/Me, 2000/XP)
3. Палитра (может отсутствовать)
4. Само изображение

BITMAPFILEHEADER

Эта структура содержит информацию о типе, размере и представлении данных в файле. Размер 14 байт.

```
typedef struct tagBITMAPFILEHEADER
{
  WORD    bfType;       // смещение 0 от начала файла
  DWORD   bfSize;       // смещение 2 от начала файла, длина 4
  WORD    bfReserved1;
  WORD    bfReserved2;
  DWORD   bfOffBits;    // смещение 10 от начала файла, длина 4
} BITMAPFILEHEADER, *PBITMAPFILEHEADER;
```

Тип `WORD` должен иметь размер 16 бит, типы `DWORD` и `LONG` — 32 бита, тип `LONG` — знаковый, порядок байтов подразумевается little endian.

- `bfType` — тип файла, символы «BM» (в HEX: `0x42 0x4d`).
- `bfSize` — размер всего файла в байтах.
- `bfReserved1` и `bfReserved2` — зарезервированы, должны содержать нули.
- `bfOffBits` — содержит смещение в байтах от начала структуры `BITMAPFILEHEADER` до непосредственно битов изображения.

После заголовка файла

BITMAPINFOHEADER

Наиболее простой вариант заголовка. Приложения для Windows NT3.51 и более ранних могут использовать только эту структуру. Размер 40 байт.

```
typedef struct tagBITMAPINFOHEADER{
  DWORD  biSize;
  LONG   biWidth;
  LONG   biHeight;
  WORD   biPlanes;
  WORD   biBitCount;
  DWORD  biCompression;
  DWORD  biSizeImage;
  LONG   biXPelsPerMeter;
  LONG   biYPelsPerMeter;
```

```
  DWORD  biClrUsed;
  DWORD  biClrImportant;
} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

biSize

Размер данной структуры в байтах. Формат BMP со временем дополнялся и по значению этого поля определяется версия формата.

biWidth

Ширина изображения в пикселях. Для Win98/Me и Win2000/XP: если поле biCompression содержит BI_JPEG или BI_PNG, здесь указана ширина распакованного изображения.

biHeight

Высота изображения в пикселях. Если содержит положительное значение — изображение записано в порядке снизу-вверх (нулевой пиксель в нижнем левом углу). Если значение отрицательное — изображение записано сверху-вниз (нулевой пиксель в верхнем левом углу изображения). Поле biCompression должно содержать значение BI_RGB или BI_BITFIELDS. Такое изображение не может быть сжато.

biPlanes

Количество цветовых плоскостей и в формате BMP содержит единицу.

biBitCount

Количество бит на пиксель. Может принимать следующие значения:

- 0 — имеет смысл для Win98/Me/2000/XP. Число бит на пиксель определяет формат JPEG или PNG.
- 1 — изображение монохромное. Член bmiColors структуры BITMAPINFO содержит два элемента. Каждый бит изображения представляет один пиксель; если бит равен нулю — пиксель имеет цвет первого элемента таблицы bmiColors, иначе — цвет второго.
- 4 — шестнадцатицветное изображение. Пиксели определяются 4-х битными индексами, каждый байт изображения содержит информацию о двух пикселях — старшие 4 бита для первого, оставшиеся — для второго.
- 8 — в палитре содержится до 256 цветов, каждый байт изображения хранит индекс в палитре для одного пикселя.
- 16 — если поле biCompression содержит значение BI_RGB, файл не содержит палитры. Каждые два байта изображения хранят интенсивность красной, зелёной и синей компоненты одного пикселя. При этом старший бит не используется, на каждую компоненту отведено 5 бит: 0RRRRRGGGGGBBBBB.

Если поле biCompression содержит значение BI_BITFIELDS, палитра хранит три четырёхбайтовых значения, определяющих маску для каждой из трёх компонент цвета. Каждый пиксель изображения представлен двухбайтным значением, из которого с помощью масок извлекаются цветовые компоненты. Для WinNT/2000/XP — последовательности бит каждой компоненты должны следовать непрерывно, не перекрываясь и не пересекаясь с последовательностями других компонент. Для Win95/98/Me —

поддерживаются только следующие маски: 5-5-5, где маска синей компоненты 0x001F, зелёной 0x03E0, красной 0x7C00; и 5-6-5, где маска синей компоненты 0x001F, зелёной 0x07E0, красной 0xF800.

- 24 — палитра не используется, каждая тройка байт изображения представляет один пиксель, по байту для интенсивности синего, зелёного и красного канала соответственно.
- 32 — Если поле biCompression содержит значение BI_RGB, изображение не содержит палитры. Каждые четыре байта изображения представляют один пиксель, по байту для интенсивности синего, зелёного и красного канала соответственно. Старший байт каждой четвёрки обычно не используется, однако позволяет хранить данные альфа-канала. Если поле biCompression содержит значение BI_BITFIELDS, в палитре хранятся три четырёхбайтных цветовых маски — для красной, зелёной и синей компоненты. Каждый пиксель изображения представлен четырьмя байтами. WinNT/2000: маски компонент не должны перекрываться или пересекаться. Windows 95/98/Me: система поддерживает только один режим сжатия, полностью аналогичный режиму без компрессии BI_RGB — старший байт каждой четвёрки используется в качестве альфа-канала, следующие три отведены для синего, зелёного и красного канала соответственно: 0xAARRGGBB.

biCompression

Тип сжатия для сжатых изображений:

Значение	**Идентификатор**	**Компрессия**
0	BI_RGB	несжатое изображение
1	BI_RLE8	сжатие RLE для 8-битных изображений
2	BI_RLE4	сжатие RLE для 4-битных изображений
3	BI_BITFIELDS	изображение не сжато, палитра содержит три 4-байтные маски для красной, зелёной и синей компонент цвета. Используется для 16- и 32-битных изображений
4	BI_JPEG	Win98/Me/2000/XP: JPEG-сжатие
5	BI_PNG	Win98/Me/2000/XP: PNG-сжатие
6	BI_ALPHABITFIELDS	WinCE: изображение не сжато, палитра содержит четыре 4-байтные маски для красной, зелёной, синей и прозрачной (альфа-канал) компонент цвета. Используется для 16- и 32-битных изображений

biSizeImage

Размер изображения в байтах. Может содержать ноль для BI_RGB-изображений. Win98/Me/2000/XP: если biCompression содержит BI_JPEG или BI_PNG, biSizeImage указывает размер BI_JPEG или BI_PNG буфера

изображения.

biXPelsPerMeter

Горизонтальное разрешение в пикселях на метр для целевого устройства. Приложение может использовать это значение для выбора из группы ресурсов изображения, наиболее подходящего для текущего устройства. Для DPI 96, которое принято в Microsoft для мониторов, оно будет равно 3780 (если считать по формуле (96 / 25,4) * 1000).

biYPelsPerMeter

Вертикальное разрешение в пикселях на метр для целевого устройства.

biClrUsed

Количество используемых цветовых индексов в палитре. Если значение равно нулю — изображение использует максимально доступное количество индексов, в соответствии со значением biBitCount и методом сжатия, указанным в biCompression.
Если содержит ненулевое значение и biBitCount меньше 16, biClrUsed указывает количество цветов, к которым будет обращаться драйвер устройства или приложение. Если biBitCount больше или равен 16, biClrUsed размер палитры, используемой для оптимизации работы системных палитр. Если biBitCount равен 16 или 32, оптимальная палитра следует сразу после трёх четырёхбайтных масок.
В упакованном изображении массив пикселей следует сразу после структуры BITMAPINFO, biClrUsed должен содержать ноль, либо реальный размер палитры.

biClrImportant

Количество элементов палитры, необходимых для отображения изображения. Если содержит ноль — все индексы одинаково важны.

Структура BITMAPINFO объединяет BITMAPINFOHEADER и палитру, предоставляя полное описание размеров и цветов изображения.

Чтобы найти палитру в структуре BITMAPINFO, приложение должно использовать информацию, хранимую в biSize, следующим образом:

pColor = ((LPSTR)pBitmapInfo + (WORD)(pBitmapInfo->bmiHeader.biSize));

Растр обычно хранится в вертикально зеркальном виде. Но есть также возможность хранить растр не в вертикально зеркальном виде. Признак того, что растр в BMP не в вертикально зеркальном виде задаётся параметром biHeight.

BITMAPV4HEADER

Расширенная версия описанной выше структуры. Win NT 3.51 и более ранние должны использовать структуру BITMAPINFOHEADER. Win98/Me/2000/XP могут использовать вместо структуры BITMAPV4HEADER структуру BITMAPV5HEADER.

```
typedef struct {
  DWORD      bV4Size;
  LONG       bV4Width;
```

```
  LONG        bV4Height;
  WORD        bV4Planes;
  WORD        bV4BitCount;
  DWORD       bV4V4Compression;
  DWORD       bV4SizeImage;
  LONG        bV4XPelsPerMeter;
  LONG        bV4YPelsPerMeter;
  DWORD       bV4ClrUsed;
  DWORD       bV4ClrImportant;
  DWORD       bV4RedMask;
  DWORD       bV4GreenMask;
  DWORD       bV4BlueMask;
  DWORD       bV4AlphaMask;
  DWORD       bV4CSType;
  CIEXYZTRIPLE bV4Endpoints;
  DWORD       bV4GammaRed;
  DWORD       bV4GammaGreen;
  DWORD       bV4GammaBlue;
} BITMAPV4HEADER, *PBITMAPV4HEADER;
```

Поля от начала структуры и до bV4ClrImportant включительно имеют то же назначение, что и соответствующие поля структуры BITMAPINFOHEADER.

- bV4RedMask — цветовая маска красной компоненты каждого пикселя, используется только если bV4Compression содержит значение BI_BITFIELDS.
- bV4GreenMask — цветовая маска зелёной компоненты каждого пикселя, используется только если bV4Compression содержит значение BI_BITFIELDS.
- bV4BlueMask — цветовая маска синей компоненты каждого пикселя, используется только если bV4Compression содержит значение BI_BITFIELDS.
- bV4AlphaMask — маска, определяющая компоненту альфа-канала.
- bV4CSType — определяет цветовое пространство изображения.
- bV4Endpoints — структура CIEXYZTRIPLE, указывающая x, y и z координаты трёх цветов, которые соответствуют конечным точкам(endpoints) цветового пространства, определённого для изображения. Это поле игнорируется, если bV4CSType не содержит значения LCS_CALIBRATED_RGB.
- bV4GammaRed — кривая тона красной компоненты. Игнорируется, если bV4CSType не содержит значения LCS_CALIBRATED_RGB. Указывается в формате 16×16.
- bV4GammaGreen — кривая тона зелёной компоненты. Игнорируется, если bV4CSType не содержит значения LCS_CALIBRATED_RGB.

- bV4GammaBlue — кривая тона синей компоненты. Игнорируется, если bV4CSType не содержит значения LCS_CALIBRATED_RGB.

BITMAPV5HEADER

Win95/NT 4.0: приложения могут использовать BITMAPV4HEADER. Win NT 3.51 и более ранние должны использовать структуру BITMAPINFOHEADER.

```
typedef struct {
  DWORD       bV5Size;
  LONG        bV5Width;
  LONG        bV5Height;
  WORD        bV5Planes;
  WORD        bV5BitCount;
  DWORD       bV5Compression;
  DWORD       bV5SizeImage;
  LONG        bV5XPelsPerMeter;
  LONG        bV5YPelsPerMeter;
  DWORD       bV5ClrUsed;
  DWORD       bV5ClrImportant;
  DWORD       bV5RedMask;
  DWORD       bV5GreenMask;
  DWORD       bV5BlueMask;
  DWORD       bV5AlphaMask;
  DWORD       bV5CSType;
  CIEXYZTRIPLE bV5Endpoints;
  DWORD       bV5GammaRed;
  DWORD       bV5GammaGreen;
  DWORD       bV5GammaBlue;
  DWORD       bV5Intent;
  DWORD       bV5ProfileData;
  DWORD       bV5ProfileSize;
  DWORD       bV5Reserved;
} BITMAPV5HEADER, *PBITMAPV5HEADER;
```

Для полей от начала структуры и до bV5GammaBlue включительно будут описаны только отличия от предыдущих версий — BITMAPINFOHEADER и BITMAPV4HEADER.

- bV5CSType — определяет цветовое пространство изображения, может принимать следующие значения:

LCS_CALIBRATED_RGB
LCS_sRGB
LCS_WINDOWS_COLOR_SPACE
PROFILE_LINKED
PROFILE_EMBEDDED

- bV5Intent — может принимать следующие значения:

LCS_GM_ABS_COLORIMETRIC

LCS_GM_BUSINESS
LCS_GM_GRAPHICS
LCS_GM_IMAGES

- bV5ProfileData — смещение в байтах от начала структуры к началу данных профиля (имя файла профиля, строка состоящая исключительно из символов кодовой таблицы 1252 и заканчивающаяся нулевым байтом). Игнорируется, если bV5CSType содержит значение, отличное от PROFILE_LINKED и PROFILE_EMBEDDED.
- bV5ProfileSize — размер данных профиля в байтах.
- bV5Reserved — зарезервировано. Содержит ноль.

Палитра

Палитра может содержать последовательность четырёхбайтовых полей по числу доступных цветов (256 для 8-битного изображения). Три младшие байта каждого поля определяют интенсивность красной, зелёной и синей компоненты цвета, старший байт не используется. Каждый пиксель изображения описан в таком случае одним байтом, содержащим номер поля палитры, в котором сохранен цвет этого пикселя.

Если пиксель изображения описывается 16-битным числом, палитра может хранить три двухбайтных значения, каждое из которых определяет маску для извлечения из 16-битного пикселя красной, зелёной и синей компонент цвета.

Файл BMP может не содержать палитры, если в нём хранится несжатое полноцветное изображение.

Данные изображения

Последовательность пикселей, записанных в том или ином виде. Пиксели хранятся построчно, снизу вверх. Каждая строка изображения дополняется нулями до длины, кратной четырём байтам.

В bmp-файлах с глубиной цвета 24 бита, байты цвета каждого пикселя хранятся в порядке BGR (Blue,Green,Red)

В bmp-файлах с глубиной цвета 32 бита, байты цвета каждого пикселя хранятся в порядке BGRA (Blue,Green,Red,Alpha)

Битность изображения

В зависимости от количества представляемых цветов, на каждую точку отводится от 1 до 48 битов:

- 1 бит — монохромное изображение (два цвета).
- 2 бита — 4 возможных цвета (режимы работы CGA) (2-битовый режим официально не стандартизован, но используется).
- 4 бита — 16-цветное изображение (режимы работы EGA).
- 8 бит (1 байт) — 256 цветов, последний из режимов, поддерживавших индексированные цвета (см. ниже).
- 16 бит (2 байта) — режим HiColor, Для 5-6-5 = 65536 возможных оттенков, для 5-5-5 = 32768 возможных оттенков.
- 24 бита (3 байта) — TrueColor. В связи с тем, что 3 байта не очень хорошо соотносятся со степенями двойки (особенно при хранении данных в

памяти, где выравнивание данных по границе слова имеет значение), вместо него часто используют 32-битное изображение. В режиме TrueColor на каждый из трёх каналов (в режиме RGB) отводится по 1 байту (256 возможных значений), общее количество цветов равно 16777216.

- 32 бита (4 байта) — этот режим практически аналогичен TrueColor, четвёртый байт обычно не используется, или в нём располагается альфа-канал (прозрачность).
- 48 бит (6 байт) — редко используемый формат с повышенной точностью передачи цвета (16 бит на канал), поддерживается относительно малым количеством программ и оборудования.

Индексированные цвета

При количестве бит 1 (2 цвета), 2 (4 цвета), 4 (16 цветов) или 8 (256 цветов) на каждый пиксель, может использоваться специальный режим индексированных цветов. В этом случае число, соответствующее каждому пикселю, указывает не на цвет, а на номер цвета в палитре. Благодаря использованию палитры имеется возможность адаптировать изображение к цветам, присутствующим на изображении. В таком случае изображение ограниченно не заданными цветами, а максимальным количеством одновременно используемых цветов.

Рекомендуемая литература:

1. Евченко А. И. OpenGL и DirectX. Программирование графики (Для профессионалов), 2006г.- 183с.
2. Быков С. Ф. Алгоритм сжатия JPEG с позиции компьютерной стеганографии // Защита информации. Конфидент. — СПб.: 2000, № 3.
3. Грибунин В. Г., Оков И. Н., Туринцев И. В. Цифровая стеганография. — М.: Солон-Пресс, 2002. — 272 с.

Методические указания
к лабораторной работе №2

«Поиск простых чисел»

1. Цель работы:

Изучение методов проверки больших чисел на простоту.

2 Задачи работы:

Оценка трудоемкости методов полного перебора, теста Ферма, метода Миллера-рабина.

Поиск первого простого десятизначного числа в дробной части числа *e*.

3. Постановка задачи:

Разработать программное средство, автоматизирующее процесс проверки числа на простоту. Необходимо реализовать метод полного перебора, тест Ферма, метод Милера-Рабина. Оценить трудоемкость методов для чисел разной длины (8бит, 16бит, 32 бит, 64бит, 128 бит). С помощью разработанных методов найти первое десятизначное простое число в дробной части числа *e*.

4 Требования к отчету:

Отчет о проделанной работе должен содержать:

- название работы
- постановка задачи;
- алгоритм полного перебора, теста Ферма, теста Миллера-Рабина;
- сравнение производительности описанных методов;
- первое простое десятизначное число в дробной части числа *e* и его позицию.

Отчет представить в электронном виде.

5 Общие положения и методика выполнения задания

Тест простоты Ферма в теории чисел — это тест простоты натурального числа *n*, основанный на малой теореме Ферма.

Содержание

Если *n*— простое число, то оно удовлетворяет сравнению

$$a^{n-1} \equiv 1 \pmod{n},$$

где *n* не делит *a*.

Выполнение сравнения

$$a^{n-1} \equiv 1 \pmod{n}$$

является необходимым, но не достаточным признаком простоты числа. То есть если хотя бы для одного *a* $a^{n-1} \not\equiv 1 \pmod{n}$, то число *n* составное, в противном случае ничего сказать нельзя, хотя вероятность того, что число является простым увеличивается. Если для составного числа *n* выполняется сравнение $a^{n-1} \equiv 1 \pmod{n}$, то число *n* называют псевдопростым по базе *a*. При проверке числа на простоту тестом Ферма выбирают несколько чисел *a*. Чем больше количество *a*, для которых $a^{n-1} \equiv 1 \pmod{n}$, тем больше вероятность, что число *n* простое. Однако существуют составные числа, для которых сравнение $a^{n-1} \equiv 1 \pmod{n}$выполняется для всех *a* взаимно простых с *n* — это числа Кармайкла. Чисел Кармайкла — бесконечное множество, наименьшее число Кармайкла — 561. Тем не менее, тест Ферма довольно эффективен для обнаружения составных чисел.

Тест Миллера — Рабина — вероятностный полиномиальный тест простоты. Тест Миллера — Рабина позволяет эффективно определять, является ли данное число составным. Однако, с его помощью нельзя строго доказать простоту числа. Тем не менее тест Миллера — Рабина часто используется в криптографии для получения больших случайных простых чисел.

Алгоритм был разработан Гари Миллером в 1976 году и модифицирован Майклом Рабином в 1980 году.

Пусть m— нечётное число большее 1. Число $m-1$однозначно представляется в виде $m-1=2^s \cdot t$, где tнечётно. Целое число a, $1<a<m$, называется **свидетелем простоты** числа m, если выполняется одно из условий:

- $a^t \equiv 1 \pmod{m}$

или

- существует целое число k, $0 \le k < s$, такое, что $a^{2^k t} \equiv m-1 \pmod{m}$.

Теорема Рабина утверждает, что составное нечётное число mимеет не более $\varphi(m)/4$различных свидетелей простоты, где $\varphi(m)$— функция Эйлера.

Алгоритм Миллера — Рабина параметризуется количеством раундов r. Рекомендуется брать r порядка величины $\log_2(m)$, где m — проверяемое число.

Для данного m находятся такие целое число s и целое нечётное число t, что $m-1=2^s t$. Выбирается случайное число a, $1 < a < m$. Если a не является свидетелем простоты числа m, то выдается ответ *«m составное»*, и алгоритм завершается. Иначе, выбирается новое случайное число a и процедура проверки повторяется. После нахождения r свидетелей простоты, выдается ответ *«m, вероятно, простое»*, и алгоритм завершается.

Алгоритм может быть записан на псевдокоде следующим образом:

Ввод: $m > 2$, нечётное натуральное число, которое необходимо проверить на простоту;
r — количество раундов.
Вывод: *составное*, означает, что m является составным числом;
вероятно простое, означает, что m с высокой вероятностью является простым числом.
Представить $m - 1$ в виде $2^s \cdot t$, где t нечётно, можно сделать последовательным делением m - 1 на 2.
<u>цикл</u> A: повторить r раз:
  Выбрать случайное целое число a в отрезке $[2, m-2]$
  $x \leftarrow a^t \bmod m$
  <u>если</u> $x = 1$ или $x = m - 1$, <u>то</u> перейти на следующую итерацию цикла A
  <u>цикл</u> B: повторить $s - 1$ раз
    $x \leftarrow x^2 \bmod m$
    <u>если</u> $x = 1$, <u>то</u> <u>вернуть</u> *составное*
    <u>если</u> $x = m - 1$, <u>то</u> перейти на следующую итерацию цикла A
  <u>вернуть</u> *составное*
<u>вернуть</u> *вероятно простое*

Из теоремы Рабина следует, что если r случайно выбранных чисел оказались свидетелями простоты числа m, то вероятность того, что m составное, не превосходит 4^{-r}.

<u>Рекомендуемая литература:</u>

1. Василенко О. Н. Теоретико-числовые алгоритмы в криптографии, МЦНМО, 2003
2. Ю. В. Нестеренко. Введение в криптографию. - Питер, 2001. - 288 с.
3. С. Б. Гашков Упрощенное обоснование вероятностного теста Миллера — Рабина для проверки простоты чисел.

Методические указания
к лабораторной работе №3

«Шифры замены и перестановки»

1. Цель работы:
Изучение шифров замены и перестановки.

2 Задачи работы:
Реализация метода замены.
Оценка криптостойкости шифров замены.
Реализация метода маршрутной перестановки.
Оценка криптостойкости метода перестановки.

3. Постановка задачи:
Разработать программное средство, осуществляющее шифрование открытого текста методом Виженера (с автоключом) и маршрутной перестановки. Составление частотных характеристик открытых текстов, оценка криптостойкости шифров с помощью полученных частотных характеристик.

4 Требования к отчету:
Отчет о проделанной работе должен содержать:
- название работы
- постановка задачи;
- алгоритм Виженера с автоключом;
- алгоритм маршрутной перестановки;
- частотные характеристики открытого текста;
-оценка криптостойкости шифров.
Отчет представить в электронном виде.

5 Общие положения и методика выполнения задания

В шифре Виженера ключ задается набором из d букв. Такие наборы подписываются с повторением под сообщением и полученные две последовательности складываются по модулю <длина алфавита> (каждая буква рассматриваемого алфавита нумеруется от А = 0 до Я = 30).

Таким образом,

$Yi = Xi+Ki \pmod{26}$,

Например, с помощью ключа «ключ» получаем

М	**А**	**М**	**А**	**М**	**Ы**	**Л**	**А**	**Р**	**А**	**М**	**У**
11	0	11	0	11	26	10	0	15	0	11	18
к	**л**	**ю**	**ч**	**к**	**л**	**ю**	**ч**	**к**	**л**	**ю**	**ч**
9	10	29	22	9	10	29	22	9	10	29	22
20	10	40	22	20	36	39	22	24	10	40	40
20	10	10	22	20	6	9	22	24	10	10	10
Х	**Л**	**Л**	**Ч**	**Х**	**Ж**	**К**	**Ч**	**Щ**	**Л**	**Л**	**Л**

Расшифрование Xi=Yi-Ki (+ при необходимости длина алфавита)

Шифр Виженера с периодом 1 называется шифром Цезаря.
Он представляет собой простую подстановку, в которой каждая буква сообщения M сдвигается вперед на фиксированное число мест по алфавиту. Это число и является ключом; оно может быть любым от 0 до 25. Так называемый шифр Бофора (Beaufort) и видоизмененный шифр Бофора подобны шифру Виженера. В них сообщения зашифровываются с помощью равенств

ei = ki - mi (mod 26)

и

ei = mi - ki (mod 26)

соответственно. Шифр Бофора с периодом 1 называется обратным шифром Цезаря.

Шифр с автоключом.

Шифр типа Виженера, в котором или само сообщение или результирующая криптограмма используются в качестве "ключа", называется шифром с автоключом. Шифрование начинается с помощью "первичного ключа" (который является настоящим ключом в нашем смысле) и продолжается с помощью сообщения или криптограммы, смещенной на длину первичного ключа, как в указанном ниже примере, где первичным ключом является набор букв COMET. В качестве "ключа" используется сообщение:

Сообщение S E N D S U P P L I E S ...

Ключ C O M E T S E N D S U P ...

Криптограмма U S Z H L M T C O A Y H ...

Если в качестве "ключа" использовать криптограмму, то получится

Сообщение	S	E	N	D	S	U	P	P	L	I	E	S
...												
Ключ	C	O	M	E	T	U	S	Z	H	L	O	H ...
Криптограмма	U	S	Z	H	L	O	H	O	S	T	T	
S ...												

Шифр, преобразования из которого изменяют только порядок следования символов исходного текста, но не изменяют их самих, называется шифром перестановки (ШП).

Широкое распространение получили шифры перестановки, использующие некоторую геометрическую фигуру. Преобразования из этого шифра состоят в том, что в фигуру исходный текст вписывается по ходу одного ``маршрута", а затем по ходу другого выписывается с нее. Такой шифр называют маршрутной перестановкой. Например, можно вписывать исходное сообщение в прямоугольную таблицу, выбрав такой маршрут: по горизонтали, начиная с левого верхнего угла поочередно слева направо и справа налево. Выписывать же сообщение будем по другому маршруту: по вертикали, начиная с верхнего правого угла и двигаясь поочередно сверху вниз и снизу вверх.

Зашифруем, например, указанным способом фразу:

ПРИМЕРМАРШРУТНОЙПЕРЕСТАНОВКИ

используя прямоугольник размера :

П	Р	И	М	Е	Р	М
Н	Т	У	Р	Ш	Р	А
О	Й	П	Е	Р	Е	С
И	К	В	О	Н	А	Т

Зашифрованная фраза выглядит так:

МАСТАЕРРЕШРНОЕРМИУПВКЙТРПНОИ

Теоретически маршруты могут быть значительно более изощренными, однако запутанность маршрутов усложняет использование таких шифров.

Рекомендуемая литература:

1. В. В. Ященко. Введение в криптографию. - Питер, 2001. - 288 с.
2. http://www.cryptography.ru/

Методические указания
к лабораторной работе №4

«Симметричные шифры»

1. Цель работы:
Изучение методов блочного симметричного шифрования

2 Задачи работы:
Разработка алгоритма шифрования.
Выбор размера блока.
Выбор размера ключа.
Выбор количества раундов.
Выбор режима шифрования.
Оценка криптостойкости разработанного шифра.

3. Постановка задачи:
Разработать программное средство, автоматизирующее процесс шифрования открытого текста с помощью блочного симметричного метода. Открытый текст, ключ, вектор инициализации загружается из указанного пользователем файла, закрытый текст сохраняется в файл.

4 Требования к отчету:
Отчет о проделанной работе должен содержать:
- название работы
- постановка задачи;
- алгоритм блочного симметричного шифра;
- руководство пользователя;
- корреляционные зависимости между битами входного и выходного файла.

Отчет представить в электронном виде.

Индивидуальные варианты

№ варианта	Вид сети Фейштеля	Режим
1	1ая модификация	OFB
2	2ая модификация	CFB
3	3ая модификация	CBC
4	1ая модификация	CFB
5	2ая модификация	OFB
6	3ая модификация	CFB
7	1ая модификация	CBC
8	2ая модификация	CBC
9	3ая модификация	OFB

5 Общие положения и методика выполнения задания

Существует модификация сети Фейштеля для большего числа ветвей, которую применяют гораздо чаще. Это в первую очередь связано с тем, что при больших размерах кодируемых блоков (128 и более бит) становится неудобно работать с математическими функциями по модулю 64 и выше. Как известно, основные единицы информации обрабатываемые процессорами на сегодняшний день – это байт и двойное машинное слово 32 бита. Поэтому все чаще и чаще в блочных криптоалгоритмах встречается сеть Фейштеля с 4-мя ветвями. Различные варианты сетей Фейштеля приведены на рисунке 3.5.

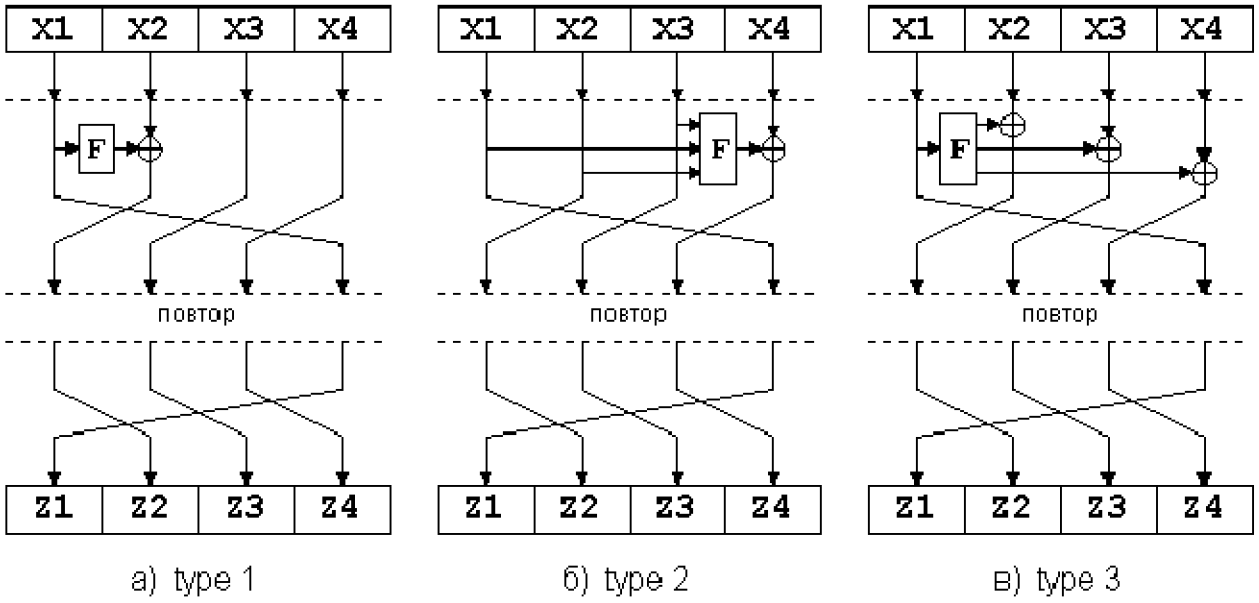


Рисунок 3.5 – Модификация сети Фейштеля

Сеть Фейштеля надежно зарекомендовала себя как криптостойкая схема произведения преобразований, и ее можно найти практически в любом современном блочном шифре. Криптостойкость блочного шифра, использующего сеть Фейштеля, определяется на 95% функцией F и правилом вычисления V_i из ключа. Эти функции и являются объектом все новых и новых исследований специалистов в области криптографии.

- Режим сцепления блоков (CBC — Cipher Block Chaining) (см. Рис.8). Каждый очередной блок C_ii>=1, перед зашифровыванием складывается по модулю 2 со следующим блоком открытого текста M_{i+1}. Вектор C_0 — начальный вектор, он меняется ежедневно и хранится в секрете.

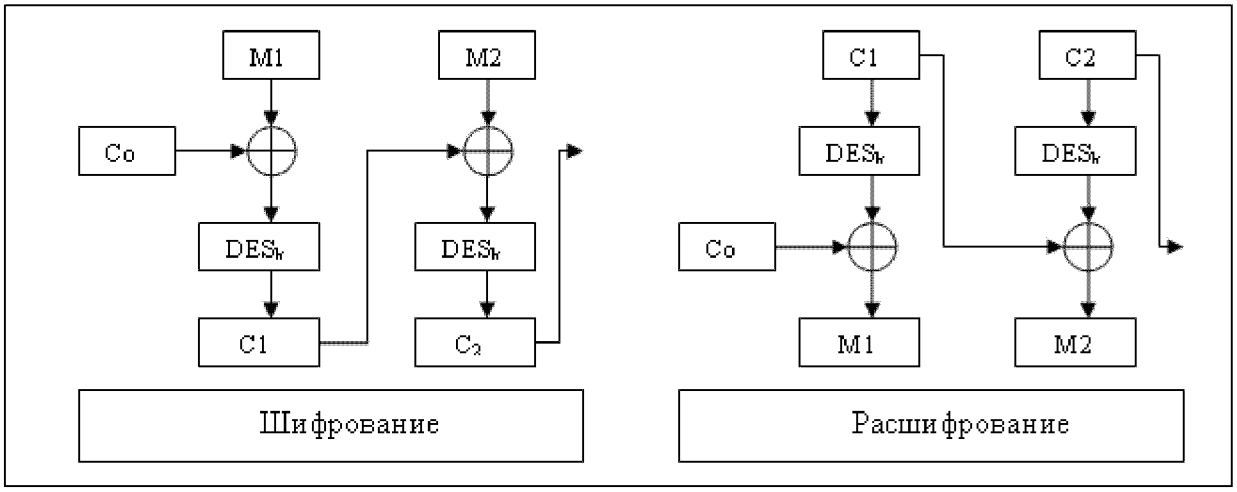


Рис.8 Режим сцепления блоков — CBC

- Режим обратной связи по шифротексту (англ. *Cipher Feed Back*) (см. Рис.9). В режиме CFB вырабатывается блочная «гамма» $Z_0, Z_1, ..Z_i = DES_k(C_{i-1})C_i = M_i \oplus Z_i.$

Начальный вектор C_0является синхропосылкой и предназначен для того, чтобы разные наборы данных шифровались по-разному с использованием одного и того же секретного ключа. Синхропосылка посылается получателю в открытом виде вместе с зашифрованным файлом.

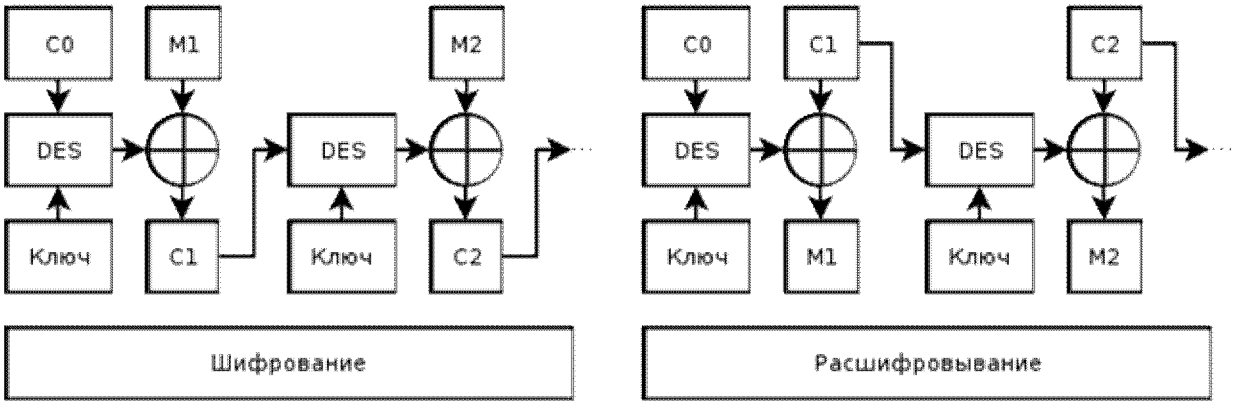


Рис.9 Режим обратной связи по шифротексту — CFB

- Режим обратной связи по выходу (OFB — Output Feed Back) (см. Рис.10). В режиме OFB вырабатывается блочная «гамма» $Z_0, Z_1, ..Z_i = DES_k(Z_{i-1})C_i = M_i \oplus Z_i,$ i>=1

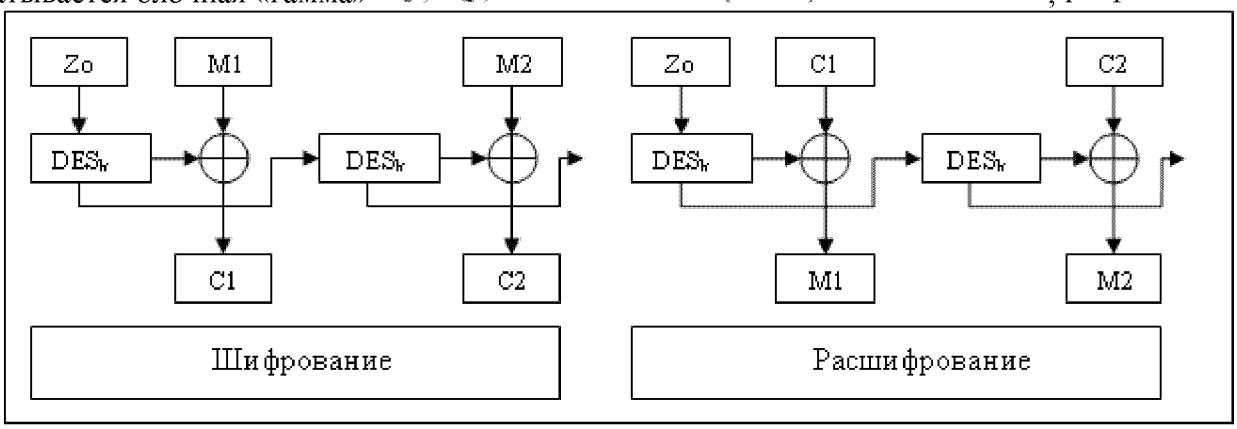


Рис.10 Режим обратной связи по выходу — OFB

Рекомендуемая литература:

1. А. П. Алферов, А. Ю. Зубов, А. С. Кузьмин, А. В. Черемушкин Основы криптографии.
2. Защита информации в компьютерных системах и сетях. Под ред. В.Ф.Шаньгина.-М.:Радио и связь, 1999.-328с.
3. Введение в криптографию. Под общ. ред. В.В.Ященко.-М.: МЦНМО: "ЧеРо", 2000.

Методические указания
к лабораторной работе №5

«Ассимметричные шифры»

1. Цель работы:
Изучение методов блочного ассимметричного шифрования

2 Задачи работы:
Выбор метода проверки числа на простоту.
Генерация ключей.
Реализация шифрования и расшифрования методом RSA.
Подбор закрытых ключей с помощью известных секретов.

3. Постановка задачи:
Разработать программное средство, автоматизирующее процесс шифрования открытого текста с помощью метода RSA. Открытый текст, ключи, загружается из указанного пользователем файла, закрытый текст сохраняется в файл. Нахождение закрытого ключа при известном значении функции Эйлера и при малых значения шифрующей экспоненты.

4 Требования к отчету:
Отчет о проделанной работе должен содержать:
- название работы
- постановка задачи;
- выбор алгоритма проверки числа на простоту;
- выбор размера p и q;
- обоснование выбора шифрующей экспоненты.
- руководство пользователя.
Отчет представить в электронном виде.

5 Общие положения и методика выполнения задания

Алгоритм RSA стоит у истоков асимметричной криптографии. Он был предложен тремя исследователями-математиками Рональдом Ривестом (R. Rivest), Ади Шамиром (A. Shamir) и Леонардом Адльманом (L. Adleman) в 1977-78 годах.

Первым этапом любого асимметричного алгоритма является создание пары ключей: открытого и закрытого и распространение открытого ключа "по всему миру". Для алгоритма RSA этап создания ключей состоит из следующих операций:

1. Выбираются два простых (!) числа p и q.
2. Вычисляется их произведение $n(=p*q)$.

3. Выбирается произвольное число e ($e<n$), такое, что НОД(e, (p-1)(q-1))=1, то есть e должно быть взаимно простым с числом (p-1)(q-1).

4. Методом Евклида решается в целых числах (!) уравнение $e*d+(p-1)(q-1)*y=1$. Здесь неизвестными являются переменные d и y – метод Евклида как раз и находит множество пар (d, y), каждая из которых является решением уравнения в целых числах.

5. Два числа (e, n) – публикуются как открытый ключ.

6. Число d хранится в строжайшем секрете – это и есть закрытый ключ, который позволит читать все послания, зашифрованные с помощью пары чисел (e, n).

Как же производится собственно шифрование с помощью этих чисел:

1. Отправитель разбивает свое сообщение на блоки, равные $k=[\log_2(n)]$ бит, где квадратные скобки обозначают взятие целой части от дробного числа.

2. Подобный блок, как Вы знаете, может быть интерпретирован как число из диапазона (0; 2^k-1). Для каждого такого числа (назовем его m_i) вычисляется выражение $c_i=((m_i)^e)\mathrm{mod}\ n$. Блоки c_i и есть зашифрованное сообщение Их можно спокойно передавать по открытому каналу, поскольку операция возведения в степень по модулю простого числа, является необратимой математической задачей. Обратная ей задача носит название "логарифмирование в конечном поле" и является на несколько порядков более сложной задачей. То есть даже если злоумышленник знает числа e и n, то по c_i прочесть исходные сообщения m_i он не может никак, кроме как полным перебором m_i.

А вот на приемной стороне процесс дешифрования все же возможен, и поможет нам в этом хранимое в секрете число d. Достаточно давно была доказана теорема Эйлера, частный случай которой утверждает, что если число n представимо в виде двух простых чисел p и q, то для любого x имеет место равенство $(x^{(p-1)(q-1)})\mathrm{mod}\ n=1$. Для дешифрования RSA-сообщений воспользуемся этой формулой. Возведем обе ее части в степень ($-y$):

$(x^{(-y)(p-1)(q-1)})\mathrm{mod}\ n=1^{(-y)}=1$. Теперь умножим обе ее части на x:

$(x^{(-y)(p-1)(q-1)+1})\mathrm{mod}\ n=1*x=x$.

А теперь вспомним как мы создавали открытый и закрытый ключи. Мы подбирали с помощью алгоритма Евклида d такое, что $e*d+(p-1)(q-1)*y=1$, то есть $e*d=(-y)(p-1)(q-1)+1$. А следовательно в последнем выражении предыдущего абзаца мы можем заменить показатель степени на число ($e*d$). Получаем $(x^{e*d})\mathrm{mod}\ n=x$. То есть для того чтобы прочесть сообщение $c_i=((m_i)^e)\mathrm{mod}\ n$ достаточно возвести его в степень d по модулю m:

$((c_i)^d)\mathrm{mod}\ n=((m_i)^{e*d})\mathrm{mod}\ n=m_i$.

На самом деле операции возведения в степень больших чисел достаточно трудоемки для современных процессоров, даже если они производятся по оптимизированным по времени алгоритмам. Поэтому обычно весь текст сообщения кодируется обычным блочным шифром (намного более быстрым), но с использованием ключа сеанса, а вот сам ключ сеанса шифруется как раз асимметричным алгоритмом с помощью открытого ключа получателя и

помещается в начало файла.

Рекомендуемая литература:

3. А. П. Алферов, А. Ю. Зубов, А. С. Кузьмин, А. В. Черемушкин Основы криптографии.
4. Защита информации в компьютерных системах и сетях. Под ред. В.Ф.Шаньгина.-М.:Радио и связь, 1999.-328с.
5. Введение в криптографию. Под общ. ред. В.В.Ященко.-М.: МЦНМО: "ЧеРо", 2000.

Методические указания
к лабораторной работе №6

«Анализ сетевого трафика»

1. Цель работы:
Изучение инструментов для «прослушивания» сетевого трафика

2 Задачи работы:
Перевод сетевой карты в «беспорядочный режим».
Захват сетевых пакетов, предназначенных для других узлов.
Анализ поля данных захваченных сетевых пакетов.

3. Постановка задачи:
Разработать программное средство, предназначенное для захвата сетевых пакетов, передаваемых в пределах сегмента локальной сети. ПС должно анализировать данные, передаваемые в пакетах и сохранять информацию, которая соответствует заданному шаблону.

4 Требования к отчету:
Отчет о проделанной работе должен содержать:
- название работы
- постановка задачи;
- описание библиотеки WinPCAP;
- руководство пользователя.
Отчет представить в электронном виде.

5 Общие положения и методика выполнения задания

Библиотека Pcap (Packet Capture) позволяет создавать программы анализа сетевых данных, поступающих на сетевую карту компьютера. Примером программного обеспечения, использующего библиотеку Pcap, служит программа Wireshark. Разнообразные программы мониторинга и тестирования сети, снифферы используют эту библиотеку. Она предназначена для использования совместно с языками C/C++, а для работы с библиотекой на других языках, таких как Java, .NET, используют оболочки. Для Unix-подобных систем это библиотека libpcap, а для Microsoft Windows - WinPcap. Программное обеспечение сетевого мониторинга может использовать libpcap или WinPcap, чтобы захватить пакеты, путешествующие по сети, и (в более новых версиях) для передачи пакетов в сети. Libpcap и WinPcap также поддерживают сохранение захваченных пакетов в файл и чтение файлов, содержащих сохранённые пакеты. Программы, написанные на основе libpcap или

WinPcap, могут захватить сетевой трафик, анализировать его. Файл захваченного трафика сохраняется в формате, понятном для приложений, использующих Pcap.

WinPcap состоит из:

- драйвера для Windows 95/98/Me, и для семейства Windows NT (Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, etc.), которые используют NDIS для чтения пакетов, которые получает сетевая карта;
- Низкоуровневые библиотеки для взаимодействия с драйверами сетевых интерфейсов.

Недостатки - работает далеко не со всеми нестандартными адаптерами (Wi-Fi карточками, VPN и т.д.).

Библиотека packet.dll представляет собой динамически загружаемую библиотеку, с помощью которой приложение пользователя взаимодействует с драйвером захвата пакетов. Функции библиотеки предназначены для упрощения процесса взаимодействия с драйвером и обеспечивают выполнение таких операций, как получение дескрипторов сетевых адаптеров, прием и передачу пакетов по сети, установку буферов и фильтров драйвера и т.д. Существует две версии библиотеки – для Windows 95/98/ME и Windows NT/2000, полностью совместимых друг с другом.

Таким образом, используя packet.dll, можно работать на всех версиях Windows без внесения каких-либо изменений в исходный код программ. Эта возможность позволила создать единую версию Libpcap для всех версий Windows.

Структуры данных

В библиотеке packet.dll описаны и используются следующие структуры данных:

1. Структура **`PACKET`** описывает принимаемый или передаваемый пакет. Состоит из следующих полей:
    - `OVERLAPPED OverLapped` – структура, описанная в DDK Windows, используется для поддержки синхронных вызовов драйвера
    - `PVOID Buffer` – указатель на буфер, содержащий пакет;
    - `UINT Length` – размер буфера;

- `PVOID Next` – указатель на следующий пакет;
- `UINT ulBytesReceived` – размер части буфера, содержащей «верные» данные;
- `BOOLEAN bIoComplete` – показывает, содержит ли буфер «верные» данные после асинхронного вызова.

2. Структура **ADAPTER** содержит описание сетевого адаптера:
   - `HANDLE hFile` – указатель на дескриптор драйвера адаптера;
   - `TCHAR SymbolicLink` – строка, содержащая имя сетевого адаптера, открытого в данный момент.
3. Структура **PACKET_OID_DATA** используется для взаимодействия с сетевым адаптером при помощи OID-запросов и установки операций:
   - `ULONG Oid` – числовой идентификатор, определяющий тип запроса/установки операции, выполняемой адаптером при помощи функции `PacketRequest` (см. далее);
   - `ULONG Length` – длина поля Data;
   - `UCHAR Data` – данные, передаваемые адаптеру или принимаемые от него.
4. Структура **bpf_insn** содержит одиночную машинную инструкцию интерпретатора BPF, и используется для передачи программы фильтра драйверу:
   - `USHORT code` – содержит тип инструкции и режимы адресации;
   - `UCHAR jt, UCHAR jf` – содержат смещение к следующей инструкции при необходимости условного перехода по результатам выполнения текущей инструкции «истина» (jt) или «ложь» (jf);
   - `int k` – зарезервировано.
5. Структура **bpf_program** является указателем на программу BPF-фильтра и используется функцией `PacketSetBPF` для установки фильтра в драйвере:
   - `UINT bf_len` – размер программы-фильтра;
   - `struct bpf_insn *bpf_insns` – указатель на первую инструкцию программы.
6. Структура **bpf_hdr** описывает заголовок, используемый драйвером при передаче принятого пакета приложению:
   - `struct timeval`:

     `tv_sec` – дата захвата в стандартном формате UNIX

(число секунд, начиная с 1/1/1970);

`tv_usec` – микросекунды захвата;

- `UINT bh_caplen` – длина снимка (захваченной порции данных);
- `UINT bh_datalen` – реальная длина захваченного пакета;
- `USHORT bh_hdrlen` – размер структуры `bpf_hdr`.

7. Структура **`bpf_stat`** используется для получения статистической информации о текущей сессии:
   - `UINT bs_recv` – число пакетов, принятых адаптером с момента начала сессии;
   - `UINT bs_drop` – число потерянных пакетов с момента начала сессии.
8. Структура **`NetType`** используется функцией `PacketGetNetType` для получения информации о типе текущего адаптера:
   - `UINT LinkType` – тип текущего сетевого адаптера;
   - `UINT LinkSpeed` – скорость адаптера в битах в секунду.

Первые две структуры используются только драйвером, тогда как остальные были заданы в библиотеке Libpcap. Эта группа структур используется для выполнения таких операций, как установка фильтра или интерпретация данных, поступающих от драйвера. На самом деле для взаимодействия с приложениями драйвер использует синтаксис, схожий с синтаксисом BPF, поэтому формат этих структур одинаков.

Функции библиотеки packet.dll

Библиотека packet.dll предоставляет набор функций, которые позволяют принять или отправить пакет произвольной структуры, запросить или установить параметры сетевого адаптера, получить дескрипторы динамически размещаемых структур типа PACKET, установить или снять BPF-фильтр, изменить размер буфера драйвера и получить статистическую информацию о текущей сессии.

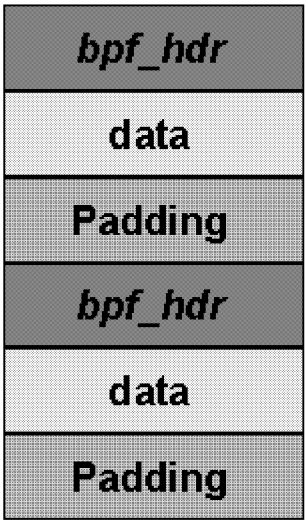
Имеются следующие функции:

- `ULONG PacketGetAdapterNames (PTSTR pStr, PULONG BufferSize)` – предназначена для получения информации об адаптерах, установленных в системе. Функция опрашивает регистр ОС, производит OID-вызовы драйвера пакетов и записывает имена установленных сетевых адаптеров и их описание в заданный пользователем буфер `pStr`. `BufferSize` – размер этого буфера. Формат данных, записываемых в буфер, отличен для версий Windows 95/98 и

WindowsNT/2000, из-за разницы в кодировках строк у этих ОС (Windows 95/98 использует кодировку ASCII, Windows NT/2000 – UNICODE).

- LPADAPTER PacketOpenAdapter (LPSTR AdapterName) – предназначена для инициализации адаптера. Функции передается имя адаптера в качестве аргумента AdapterName (получено с помощью PacketGetAdapterNames), результатом функции является указатель на структуру ADAPTER открытого адаптера.
- VOID PacketCloseAdapter (LPADAPTER lpAdapter) – высвобождает структуру ADAPTER, связанную с указателем lpAdapter, и закрывает адаптер, связанный с ней.
- LPPACKET PacketAllocatePacket (void) – определяет положение структуры PACKET, инициализированной функцией PacketInitPacket, и возвращает указатель на нее.
- VOID PacketInitPacket (LPPACKET lpPacket, PVOID Buffer, UINT Length) – инициализирует структуру PACKET и имеет следующие аргументы:
  - lpPacket – указатель на инициализируемую структуру;
  - Buffer – указатель на буфер, задаваемый пользователем и содержащий данные пакета;
  - Length – длина буфера – максимальный размер данных, которые могут быть переданы драйвером приложению за один сеанс чтения.
- VOID PacketFreePacket (LPPACKET lpPacket) – высвобождает структуру PCAKET, связанную с указателем lpPacket.
- VOID PacketReceivePacket (lpAdapter AdapterObject, LPPACKET lpPacket, BOOLEAN Sync) – выполняет захват группы пакетов, и имеет следующие аргументы:
  - AdapterObject – указатель на структуру ADAPTER, определяющую адаптер, который будет задействован в текущей сессии;
  - lpPacket – указатель на структуру PACKET, используемую для записи принятых пакетов;
  - Sync – флаг, определяющий режим выполнения операции. Если выбран синхронный режим (True), функция блокирует программу до завершения операции. Если выбран асинхронный режим (False), блокировки не происходит. В последнем случае необходимо использовать функцию PacketWaitPaket для корректного выполнения операции.

Число принятых пакетов зависит от количества пакетов, сохраненных в буфере драйвера, размера этих пакетов и размера буфера, связанного со структурой `lpPacket`. Ниже показан формат передачи данных приложению драйвером.



Пакеты сохраняются в буфере структуры `lpPacket`. Каждый пакет имеет трейлер, состоящий из структуры `bpf_hdr` и содержащий информацию о длине пакета и времени его приема. Поле `Padding` используется для выравнивания данных в буфере. Поля `bf_datalen` и `bf_hdrlen` структуры `bpf_hdr` используются для извлечения пакетов из буфера. Заметим, что Pcap извлекает каждый пакет до того, как передать его приложению.

- `BOOLEAN PacketWaitPacket (LPADAPTER AdapterObject, LPPACKET lpPacket)` – используется для корректного завершения операции ввода/вывода драйвера захвата пакетов. Она является блокирующей в том случае, если драйвер выполняет операцию ввода/вывода. Функция возвращает значение True, если операция завершена успешно, в противном случае – False. Используя функцию `GETLASTERROR`, можно получить код возникшей ошибки.

- `BOOLEAN PacketSendPacket (LPADAPTER AdapterObject, LPPACKET lpPacket, BOOLEAN Sync)` – позволяет передать сформированный пользователем пакет ПРОИЗВОЛЬНОЙ СТРУКТУРЫ в сеть через адаптер, заданный переменной `AdapterObject`. При этом пользователь программным образом создает заголовок пакета, заполняет его данными и отправляет его в сеть «как есть». Формировать структуру `bpf_hdr` перед заголовком отправляемого пакета не нужно. Также нет необходимости рассчитывать CRC пакета, поскольку она будет автоматически рассчитана сетевым интерфейсом и помещена в конце блока данных. Функция имеет те же аргументы, что и

PacketReceivePacket.

Возможности данной функции дополняет функция PacketSetNumWrites, которая устанавливает число повторов передачи одного пакета при вызове функции PacketSendPacket.

- BOOLEAN PacketResetAdapter (LPADAPTER AdapterObject) - сбрасывает адаптер, указанный в качестве аргумента.
- BOOLEAN PacketSetHwFilter (LPADAPTER AdapterObject, ULONG Filter) – устанавливает аппаратный (hardware) фильтр входящих пакетов. Константы, с помощью которых задается фильтр, объявлены в файле *ntddndis.h*. В качестве аргументов функции задается адаптер, на который устанавливается фильтр, и идентификатор фильтра. Функция возвращает значение True, если операция выполнена успешно. Ниже перечислены наиболее часто используемые фильтры:
  - *NDIS_PACKET_TYPE_PROMISCUOUS*: каждый входящий пакет принимается адаптером;
  - *NDIS_PACKET_TYPE_DIRECTED*: принимаются пакеты, предназначенные для данной рабочей станции;
  - *NDIS_PACKET_TYPE_BROADCAST*: принимаются только широковещательные запросы;
  - *NDIS_PACKET_TYPE_MULTICAST*: принимаются пакеты, предназначенные группе, которой принадлежит рабочая станция;
  - *NDIS_PACKET_TYPE_ALL_MULTICAST*: принимаются пакеты любой группы.
- BOOLEAN PacketRequest (LPADAPTER AdapterObject, BOOLEAN Set, PACKET_OID_DATA OidData) – предназначена для выполнения запроса/установки параметров адаптера AdapterObject. Эта функция используется для получения значений различных параметров сетевого адаптера (размер внутреннего буфера, скорость соединения, значение счетчика пакетов и др.) или их изменения. Второй аргумент определяет тип операции (Set=1 – установка параметра, Set=0 – запрос на получение значения параметра). Третий аргумент – указатель на структуру PACKET_OID_DATA, определяющую параметр адаптера. Функция возвращает True, если операция была выполнена без ошибок.
- BOOLEAN PacketSetBuff (LPADAPTER AdapterObject, int dim) - устанавливает новый размер буфера драйвера, связанного с адаптером AdapterObject. dim – новый размер буфера. Функция возвращает True, если операция была выполнена успешно, False – если для выполнения операции недостаточно памяти. При установке нового размера буфера все данные, находящиеся в нем, стираются.

- BOOLEAN PacketSetBpf (LPADAPTER AdapterObject, struct bpf_program *fp) – связывает новый BPF-фильтр с адаптером AdapterObject. Фильтр, указанный *fp, представляет собой набор инструкций, которые будут выполняться после приема каждого входящего пакета. Функция возвращает значение True, если фильтр установлен успешно, False – если программа-фильтр не принята драйвером.

  Фильтр может быть создан автоматически путем вызова функции pcap_compile библиотеки **libpcap**. При этом происходит преобразование текстового фильтра (синтаксис фильтра описан в предыдущей статье) в программу BPF. Если использование функций **libpcap** нецелесообразно, имеется возможность получения исходного кода путем запуска программы WinDump с указанием текстового фильтра и ключей -d, -dd или -ddd. Полученный псевдокод можно использовать в функции PacketSetBpf.

- BOOLEAN PacketGetStats (LPADAPTER AdapterObject, struct bpf_stst *s) – с помощью этой функции программист может получить значение двух внутренних переменных драйвера:

  - число пакетов, принятых адаптером AdapterObject, с момента его открытия при помощи функции PacketOpenAdapter;
  - число пакетов, принятых адаптером, но проигнорированных ядром. Пакет игнорируется в том случае, если приложение пользователя не готово принять его и буфер, связанный с адаптером, заполнен.

  В структуру bpf_stat драйвер записывает два значения:

  - большое значение переменной bs_receive означает, что трафик сети весьма высок. Если приложению нет необходимости принимать все входящие пакеты, рекомендуется установить избирательный фильтр BPF для уменьшения числа обрабатываемых приложением пакетов.
  - если значение переменной bs_drop выше нуля, приложение пользователя не успевает обрабатывать входящие пакеты и теряет их. В этом случае рекомендуется увеличить размер буфера при помощи функции PacketSetBuff, увеличить размер буфера в структуре PACKET либо оптимизировать алгоритм приложения.

- BOOLEAN PacketGetNetType (LPADAPTER AdapterObject, NetType *type) – записывает тип адаптера AdapterObject в структуру NetType. Переменной LinkType структуры может быть присвоено одно из следующих значений:

- NdisMedium802_3: Ethernet (802.3)
- NdisMedium802_5: Token Ring (802.5)
- NdisMediumFddi: FDDI
- NdisMediumWan: WAN
- NdisMediumLocalTalk: LocalTalk
- NdisMediumDix: DIX
- NdisMediumAtm: ATM
- NdisMediumArcnetRaw: ARCNET (raw)
- NdisMediumArcnet878_2: ARCNET (878.2)
- NdisMediumWirelessWan: Различные NdisWireless *Xxx*.

Переменная `LinkSpeed` содержит скорость передачи данных по сети в битах в секунду. Функция возвращает значение True, если операция выполнена успешно.

- `BOOLEAN PacketSetReadTimeout (LPADAPTER AdapterObject, int timeout)` – устанавливает время ожидания (аргумент timeout) входящего пакета, по истечении которого все пакеты, сохраненные в буфере, будут переданы приложению.

- `BOOLEAN PacketSetMode (LPADAPTER AdapterObject, int mode)` - переводит адаптер AdapterObject в режим работы mode, имеющий два возможных значения:

  - *MODE_CAPT* – стандартный режим захвата, устанавливается по умолчанию после вызова функции `PacketOpenAdapter`;

  - *MODE_STAT* – режим статистики. В этом режиме драйвер не перехватывает пакеты, а производит лишь подсчет принятых пакетов и принятых байт. Эти значения могут быть запрошены приложением путем вызова функции `PacketReceivePacket`. Приложение имеет возможность получать данную информацию периодически по истечении времени ожидания, которое по умолчанию составляет 1 секунду, но может быть изменено на другое значение с дискретностью 1 мС при помощи функции `PacketSetReadTimeout`.

Для того, чтобы работать в режиме статистики, приложению необходимо выполнить следующие операции:

- открыть адаптер
- установить режим статистики функцией `PacketSetMode`
- установить фильтр, определяющий тип подсчитываемых пакетов, функцией `PacketSetBpf`
- установить необходимое время ожидания при помощи `PacketSetReadTimeout`
- принять результаты при помощи `PacketReceivePacket`.

При этом функция `PacketReceivePacket` возвращает число принятых пакетов и число принятых байт, удовлетворяющих требованиям установленного фильтра, за последний временной интервал. Эти значения являются 64-битными целыми числами, и снабжены заголовком, определяемым структурой `bpf_hdr`:

struct timeval **bh_tstamp**
UINT **bh_caplen**=16
UINT **bh_datalen**=16
USHORT **bh_hdrlen**=18
LARGE_INTEGER **PacketsAccepted**
LARGE_INTEGER **BytesAccepted**

Заметим, что если интерфейс работает в режиме статистики, то в буфере ядра нет необходимости, поскольку статистические значения вычисляются без сохранения в нем данных.

- `BOOLEAN PacketSetNumWrites (LPADAPTER AdapterObject, int nwrites)` – определяет число повторов одиночной операции передачи данных (аргумент `nwrites`). По умолчанию `nwrites=1`, однако при помощи этой функции значение можно изменить. Так, если установлено 5 повторов, то вместо одного пакета, отправляемого при помощи вызова `PacketReceivePacket`, будет отправлено 5 копий этого пакета.

Создание приложения с использованием библиотеки packet.dll

Метод создания приложения рассмотрим на примере программы Traffic Generator. Эта простая программа показывает, как создать пакет произвольной структуры и отправить его в сеть при помощи драйвера захвата пакетов. Напоминаем, что программа может быть откомпилирована под Win32 в среде Microsoft Visual C++ 6.0 (не забудьте предварительно установить библиотеку libpcap и выполнить настройку переменных среды для обеспечения

возможности компиляции программ, использующих эту библиотеку). Для работы программы под Win32 необходимо установить драйвера WinPCap.

Traffic Generator представляет собой имитатор сетевого трафика. В качестве исходных данных при запуске программы необходимо указать три параметра: имя используемого интерфейса, число отправляемых пакетов и их размер. Генерируемые пакеты в качестве источника имеют MAC-адрес 1:1:1:1:1:1, и адрес 2:2:2:2:2:2 в качестве приемника.

Исходный код программы Traffic Generator приведен ниже.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>

#include "..\..\Include\packet32.h" //подключаем заголовочный файл библиотеки
packet.dll

// задаем переменные
#define SIMULTANEOU_READS 10
#define MAX_ETHERNET_FRAME_SIZE 1514
#define Max_Num_Adapter 10 //максимальное число адаптеров

// Prototypes

void PrintPackets(LPPACKET lpPacket); // печать пакетов

char AdapterList[Max_Num_Adapter][1024];// список адаптеров - по 1024 байт на
адаптер



int main(int argc, char **argv)
{

 char packetbuff[5000];// буфер для данных пакета

 // указатель на структуру ADAPTER

 LPADAPTER  lpAdapter = 0;

 // указатель на структуру PACKET

 LPPACKET   lpPacket;

 int        i,npacks,Snaplen;
 DWORD      dwErrorCode;

 DWORD dwVersion;
 DWORD dwWindowsMajorVersion;

 // строки формата unicode (winnt)
 WCHAR        AdapterName[512];   //   строка,   содержащая   список   сетевых
адаптеров
 WCHAR        *temp,*temp1;

 // строки формата ascii (win95)
 char         AdapterNamea[512];   //   строка,   содержащая   список   сетевых
адаптеров     char         *tempa,*temp1a;
```

```
        int                 AdapterNum=0,Open;// номер адаптера
        ULONG        AdapterLength; // длина

        float  cpu_time;

        printf("Генератор сетевого трафика v 0.9999\nCopyright 1999 Loris Degioanni
(loris@netgroup-serv.polito.it)");
        printf("\nОтправляет группу пакетов в сеть.");

        if (argc == 1){
              printf("\n\n Параметры: tg [-i адаптер] -n число_пакетов -s размер");
              printf("\n размер в пределах от 60 до 1514\n\n");
              return -1;
        }



        AdapterNamea[0]=0; // инициализация переменной

        // получение параметров командной строки
        for(i=1;i<argc;i+=2){

              switch (argv[i] [1])
              {

              case 'i':
                    sscanf(argv[i+1],"%s",AdapterNamea);
                    break;

              case 'n':
                    sscanf(argv[i+1],"%d",&npacks);
                    break;

              case 's':
                    sscanf(argv[i+1],"%d",&Snaplen);
                    break;

              }

        }




        if(AdapterNamea[0]==0){
              // если имени адаптера при запуске не указано
              // получаем имена адаптеров, установленных в системе
              AdapterLength=1024;

              printf("Установленные адаптеры:\n");
              i=0;// номер первого адаптера

        // формат данных, возвращаемых PacketGetAdapterNames различен в Win95 и в
WinNT.
        // из-за разных кодировок
       // поэтому необходимо проверить версию ОС, на которой работает программа
              dwVersion=GetVersion();
              dwWindowsMajorVersion =  (DWORD)(LOBYTE(LOWORD(dwVersion)));
              if (!(dwVersion >= 0x80000000 && dwWindowsMajorVersion >= 4))
              {  // Windows NT
                    //получаем  список  имен  адаптеров,  записанных  в  строку
AdapterName
```

```
                PacketGetAdapterNames(AdapterName,&AdapterLength);
                temp=AdapterName;// работать будем не с AdapterName, а
                temp1=AdapterName;// с temp и с temp1
                while ((*temp!='\0')||(*(temp-1)!='\0'))// конец списка - два
 нуля
                                                // ASCII
                // имя записывается в соотв. строку массива AdapterNames
                {
                        if (*temp=='\0')// один ноль ASCII разделяет имена
адаптеров
                        {
                        // копируем имя адаптера в массив из temp1, длина имени
                        // рассчитывается при помощи temp
                                memcpy(AdapterList[i],temp1,(temp-temp1)*2);
                                temp1=temp+1;
                                i++;
                        }

                        temp++;
                }

                AdapterNum=i;// число установленных адаптеров
                for (i=0;i<AdapterNum;i++)
                        wprintf(L"\n%d-  %s\n",i+1,AdapterList[i]);//  печатаем
список
                printf("\n");

        }

        else  //windows 95
        // обработка осуществляется аналогично, за исключением того, что
        // длина имени рассчитывается для кодировки ascii
        {
                PacketGetAdapterNames(AdapterNamea,&AdapterLength);
                tempa=AdapterNamea;
                temp1a=AdapterNamea;

                while ((*tempa!='\0')||(*(tempa-1)!='\0'))
                {
                        if (*tempa=='\0')
                        {
                                memcpy(AdapterList[i],temp1a,tempa-temp1a);
                                temp1a=tempa+1;
                                i++;
                        }
                        tempa++;
                }

                AdapterNum=i;
                for (i=0;i<AdapterNum;i++)
                        printf("\n%d- %s\n",i+1,AdapterList[i]);
                printf("\n");

        }

        do
        {
                printf("Выберите номер открываемого адаптера :
");scanf("%d",&Open);
                if (Open>AdapterNum) printf("\nНомер должен быть меньше
%d",AdapterNum);
        } while (Open>AdapterNum);
```

```
		lpAdapter  =      PacketOpenAdapter(AdapterList[Open-1]);//  открываем
адаптер
		// -1, поскольку номера печатались от 1, а первый адаптер в системе
	//имеет номер 0
		// если указатель не был получен, или поле hFile содержит значение
	// INVALID_HANDLE_VALUE, открыть адаптер не удалось.
	// Код ошибки можно прочитать при помощи GetLastError
		if (!lpAdapter || (lpAdapter->hFile == INVALID_HANDLE_VALUE))
		{
			dwErrorCode=GetLastError();
			printf("Unable   to   open   the   driver,   Error   Code   :
%lx\n",dwErrorCode);


			return(-1);
		}


	}
	// если было указано имя адаптера, то обработка ошибки аналогична
	else{


		lpAdapter =  PacketOpenAdapter(AdapterNamea);


		if (!lpAdapter || (lpAdapter->hFile == INVALID_HANDLE_VALUE))
		{
			dwErrorCode=GetLastError();
			printf("Unable   to   open   the   driver,   Error   Code   :
%lx\n",dwErrorCode);


			return(-1);
		}


	}


	// устанавливаем сетевой адаптер в режим приема всех пакетов
	PacketSetHwFilter(lpAdapter,NDIS_PACKET_TYPE_PROMISCUOUS);

	// ошибка, если не был получен указатель на структуру PACKET
	if((lpPacket = PacketAllocatePacket())==NULL){
		printf("\nОшибка: невозможно обнаружить структуру LPPACKET.");
		return (-1);
	}

	// так побайтно создается пакет произвольной структуры.
	// пакет отправляется в сеть в таком виде, в каком он записан в буфере,
	// поэтому программисту нужно самому создавать заголовки и т.д.

	packetbuff[0]=1;
	packetbuff[1]=1;
	packetbuff[2]=1;
	packetbuff[3]=1;
	packetbuff[4]=1;
	packetbuff[5]=1;


	packetbuff[6]=2;
	packetbuff[7]=2;
	packetbuff[8]=2;
	packetbuff[9]=2;
	packetbuff[10]=2;
	packetbuff[11]=2;


	for(i=12;i<1514;i++){
		packetbuff[i]=i%256;
```

```
        }

       // пакет сформирован


        PacketInitPacket(lpPacket,packetbuff,Snaplen);
        // инициализация структуры PACKET в соответствии с данными
        // Snaplen - размер пакета


        // устанавливаем количество пакетов
        PacketSetNumWrites(lpAdapter,npacks);


        printf("\n\nСгенерировано %d пакетов...",npacks);


        cpu_time = clock (); // начало времени передачи


        // отправляем пакеты в синхронном режиме
        PacketSendPacket(lpAdapter,lpPacket,TRUE);


        cpu_time = (clock() - cpu_time)/CLK_TCK; //вычисляем время, затраченное на
передачу


        printf ("\n\nВремя передачи: %5.3f\n", cpu_time);
        printf ("\nСгенерировано пакетов = %d", npacks);
        printf ("\nСгенерировано байт = %d", (Snaplen+24)*npacks);
        printf ("\nСгенерировано бит = %d", (Snaplen+24)*npacks*8);
        printf       ("\nСкорость       передачи       пакетов/сек.       =       %d",
(int)((double)npacks/cpu_time));
        printf       ("\nСкорость       передачи       байт/сек.       =       %d",
(int)((double)((Snaplen+24)*npacks)/cpu_time));
        printf       ("\nСкорость       передачи       бит/сек.       =       %d",
(int)((double)((Snaplen+24)*npacks*8)/cpu_time));
        printf ("\n");


        PacketFreePacket(lpPacket);// высвобождаем структуру


        // закрываем адаптер и выходим из программы


        PacketCloseAdapter(lpAdapter);
        return (0);
```

Заключение

В заключении приведем несколько советов по созданию высокопроизводительных приложений:

1. Размер буфера драйвера и буфера структуры PACKET является очень важным параметром, определяющим быстродействие приложения. Если буфер драйвера создается библиотекой packet.dll, то его размер равен 0 (libpcap создает буфер размером 1 мегабайт), что означает очень низкую производительность. Рекомендуемый размер буфера драйвера – 512 кБ, буфера структуры PACKET – 256 кБ и выше.

2. Установите фильтр на входящие пакеты и отсеивайте пакеты, не нужные приложению. Если в данных пакетов нет необходимости, создайте фильтр, оставляющий только заголовки (первые 68 байт) пакетов.

3. При проведении статистического анализа трафика в реальном времени используйте статистический режим.

Рекомендуемая литература:

6. А. П. Алферов, А. Ю. Зубов, А. С. Кузьмин, А. В. Черемушкин Основы криптографии.
7. Защита информации в компьютерных системах и сетях. Под ред. В.Ф.Шаньгина.-М.:Радио и связь, 1999.-328с.
8. Введение в криптографию. Под общ. ред. В.В.Ященко.-М.: МЦНМО: "ЧеРо", 2000.