Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра программного обеспечения информационных технологий

А. П. Занкович

ЗАЩИТА ИНФОРМАЦИИ

Практикум
для студентов специальности I – 40 01 01
«Программное обеспечение информационных технологий»
дневной и дистанционной форм обучения

Минск 2006

УДК 681.326.7(075.8)
ББК 32.973 я 73
З-40

Р е ц е н з е н т:
доцент кафедры ЭВМ БГУИР,
кандидат технических наук В. В. Ракуш

Занкович, А. П.

З-40 Защита информации : практикум для студ. спец. I – 40 01 01 «Программное обеспечение информационных технологий» дневн. и дистанционной форм обуч. / А. П. Занкович. – Минск : БГУИР, 2006. – 39 с. : ил.

ISBN 985-488-072-9

Рассмотрены практические вопросы защиты информации в компьютерных системах – криптографическая защита данных, протоколы аутентификации сообщений и пользователей, управление доступом, атаки на переполнение буфера, стеганография и защита от нежелательной электронной корреспонденции. По каждой теме приводятся теоретические сведения, практические способы реализации алгоритмов и набор заданий разной степени сложности по вариантам.

УДК 681.326.7(075.8)
ББК 32.973 я 73

ISBN 985-488-072-9

СОДЕРЖАНИЕ

1. КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ДАННЫХ С ПОМОЩЬЮ MICROSOFT CRYPTOAPI

Структура и основные положения CryptoAPI

В настоящее время существует довольно много криптографических библиотек и интерфейсов, позволяющих защитить информацию любого прикладного приложения. Один из интерфейсов, получивший широкое распространение, – Microsoft CryptoAPI. Распространение CryptoAPI связано не только с его удобством, документированностью и доступностью. Важнейшим фактором является то, что фирма Microsoft самым активным образом интегрировала его в свои операционные системы и прикладные программы. Современные операционные системы Microsoft (Windows 2000, Windows XP, Windows ME) содержат множество криптографических подсистем различного назначения, как прикладного уровня, так и уровня ядра, и ключевую роль в реализации этих подсистем играет интерфейс CryptoAPI.

Концепция CryptoAPI подразумевает сокрытие от программиста деталей внутренней реализации криптографических алгоритмов и выполнение криптографических операций через стандартизированный интерфейс системных вызовов. Криптографические алгоритмы непосредственно реализуются внутри криптопровайдеров CSP (Cryptographic Service Provider) (рис. 1). Помимо набора стандартных криптопровайдеров каждый разработчик имеет возможность самостоятельно расширить функциональность системы CryptoAPI с помощью собственного криптопровайдера.

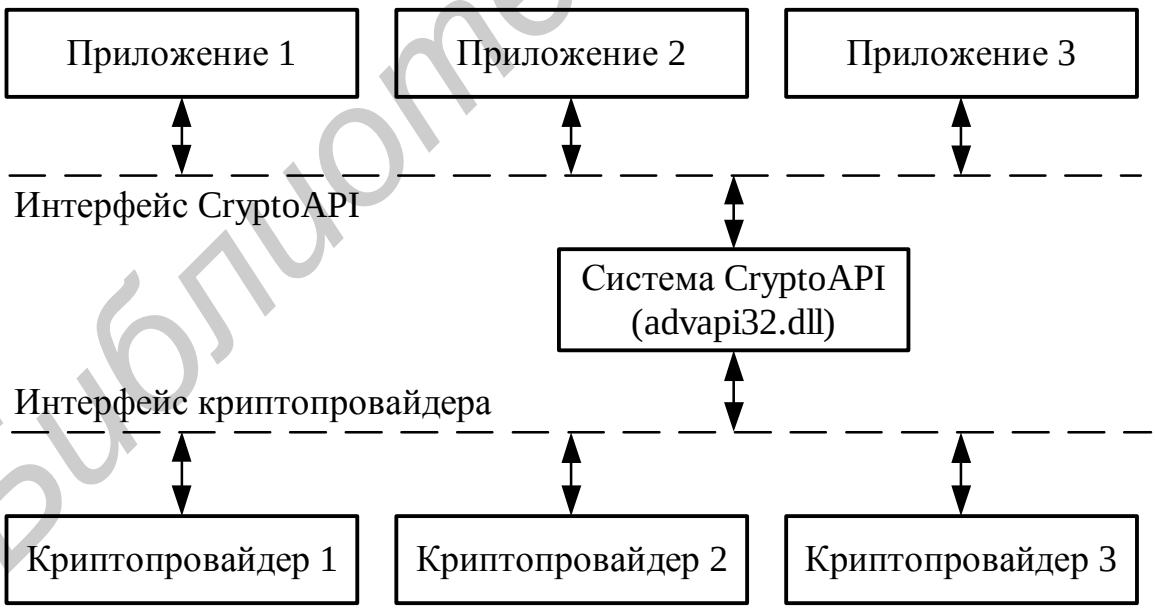

Рис. 1. Архитектура CryptoAPI

Вся архитектура CryptoAPI может быть разделена на три основные части:

1. Базовые функции.
2. Функции шифрования и для работы с сертификатами.
3. Функции для работы с сообщениями.

В группу базовых функций входят функции для выбора и подключения к криптопровайдеру, генерации и хранения ключей, обмена ключами. Сюда также входят функции управления параметрами ключа, такими как: режим сцепления блоков, инициализационный вектор и затравочное значение salt. На данный момент Microsoft поддерживает только режимы CBC и ECB сцепления блоков, хотя константы для режимов CFB и OFB уже заведены. Сейчас Microsoft разрабатывает новый продукт CNG (Cryptography API: Next Generation). Он будет доступен с выходом Windows Vista и станет заменой CryptoAPI, предоставляя еще более систематизированный подход к шифрованию данных, в том числе и поддержку режимов CFB и OFB.

К базовым функциям также можно отнести криптографическую функцию генерации случайных данных *CryptGenRandom*. Это одна из важнейших функций во всей криптосистеме, потому что она используется для генерации ключей. Вся ответственность за ее реализацию ложится на разработчиков криптопровайдера. И если он работает в паре с каким-нибудь оборудованием, она может использовать его возможности. В случае, когда криптопровайдер реализован чисто программно, данными, которыми он пользуется для генерации случайной последовательности, могут быть системное время, позиция курсора мыши на экране компьютера, некоторое состояние устройства, например буфер ввода/вывода клавиатуры, или количество выполняемых задач в операционной системе.

Основные функции первой группы: *CryptAcquireContext, CryptReleaseContext, CryptGenKey, CryptDestroyKey, CryptExportKey, CryptImportKey, CryptDeriveKey, CryptGenRandom*.

К функциям шифрования и функциям для работы с сертификатами относят все функции для хеширования данных, шифрования и дешифрования, а также функции управления сертификатами, основным назначением которых является безопасное распространение открытых ключей пользователей. Основные функции второй группы: *CryptEncrypt, CryptDecrypt, CryptCreateHash, CryptDestroyHash, CryptHashData, CryptSignHash, CertOpenStore, CertCloseStore, CertFindCertificateInStore*.

Функции для работы с сообщениями предназначены для работы с данными в стандартизованном формате PKCS #7, разработанном RSA Laboratories.

В CryptoAPI существует традиционное разделение ключей шифрования на сессионные (симметричные) и открытые ключи. Все ключи управляются и используются при помощи идентификаторов HCRYPTKEY, и приложение не всегда может получить их значения в открытом виде.

Сессионные ключи меняются от сессии к сессии, и криптопровайдер не сохраняет их на диски или другую энергонезависимую память. В том случае, когда приложению требуется в каком-либо виде получить ключ, чтобы, скажем, передать его по незащищенному каналу передачи данных или зашифровать файл и расшифровать его через некоторое время, используется функция экспортирования ключей. Сессионные ключи могут генерироваться не только случайным образом, но и по каким-нибудь данным. В последнем случае гарантируется, что один и тот же криптопровайдер будет возвращать один и тот же

ключ по одинаковым данным. Такая возможность используется для генерирования ключей по паролям.

Открытые ключи и их закрытые части для несимметричного шифрования, в отличие от сессионных ключей, хранятся криптопровайдером в контейнерах ключей в зашифрованном виде. Реализованы последние могут быть различными способами: файлы на дисках, ключи в реестре или внешние устройства, подключаемые к компьютеру. Этими ключами можно обмениваться с помощью тех же функций, что и для сессионных ключей.

Рассмотрим практическую реализацию криптографических операций с помощью функций CryptoAPI. Поскольку эти функции являются частью интерфейса программирования операционной системы Windows, они могут быть вызваны из приложения, написанного на любом языке программирования.

Криптографические провайдеры CryptoAPI

Любой сеанс работы с CryptoAPI начинается с инициализации (получения контекста), выполняемой функцией **CryptAcquireContext**. В качестве параметров эта функция принимает имя контейнера ключей, имя криптопровайдера, его тип и флаги, определяющие тип и действия с контейнером ключей и режим работы криптопровайдера:

```
BOOL WINAPI CryptAcquireContext(
  HCRYPTPROV*     phProv,          //дескриптор провайдера
  LPCTSTR         pszContainer,    //имя контейнера ключей
  LPCTSTR         pszProvider,           //имя провайдера
  DWORD           dwProvType,      //тип провайдера
  DWORD           dwFlags);        //флаги
```

Существует около семи стандартных провайдеров, предустановленных в системе. Список установленных криптопровайдеров хранится в следующем разделе реестра:

`HKLM\SOFTWARE\Microsoft\Cryptography\Defaults\Provider`

и может быть получен с помощью функции **CryptEnumProviders**. Наиболее часто используются два стандартных криптопровайдера: Microsoft Base Cryptographic Provider (**MS_DEF_PROV**) и Microsoft Enhanced Cryptographic Provider (**MS_ENHANCED_PROV**). Следует заметить, что Enhanced-провайдер присутствует только на тех машинах, где установлена поддержка 128-битного шифрования (она автоматически устанавливается вместе с Internet Explorer 6.0).

Тип провайдера определяет множество поддерживаемых им криптоалгоритмов. Например, провайдер с типом **PROV_RSA_FULL** должен поддерживать алгоритмы DES и 3DES для симметричного шифрования, SHA1 и MD5 – для вычисления криптографического хеша и алгоритм RSA – для обмена ключами и постановки подписи.

Контейнерами ключей называют защищенные области, в которые приложения могут сохранять и извлекать сгенерированные один раз ключи, обеспечивая их защиту от злоумышленника. Стандартные криптопровайдеры хранят

ключи на диске в зашифрованном виде. Контейнеры бывают двух типов – пользовательские (этот тип используется по умолчанию) и машинные (**CRYPT_MACHINE_KEYSET**). Пользовательский контейнер доступен только приложениям, выполняемым от имени владельца контейнера. Приложение может использовать такой контейнер для сохранения персональных ключей. Доступ к машинным контейнерам разрешен только администраторам. В них обычно сохраняются ключи, используемые сервисами и системными программами.

Тип контейнера задается флагом при получении контекста. Для первоначального создания контейнера нужно вызвать **CryptAcquireContext** с флагом **CRYPT_NEWKEYSET**. Для удаления контейнера требуется указать флаг **CRYPT_DELETEKEYSET**. Если приложению не требуется доступа к контейнеру ключей (например, приложение вычисляет хеш MD5), то стоит вызывать **CryptAcquireContext** с флагом **CRYPT_VERIFYCONTEXT**, передавая NULL вместо имени контейнера. Инициализацию интерфейса CryptoAPI для последующего вычисления хеша MD5 демонстрирует следующий пример:

```
HCRYPTPROV hProv;

if(!CryptAcquireContext(&hProv, NULL, MS_DEF_PROV,
  PROV_RSA_FULL, CRYPT_VERIFYCONTEXT))
  return 0;

// ...

CryptReleaseContext(hProv,0);
```

Деинициализация CryptoAPI выполняется с помощью функции **CryptReleaseContext**, единственным значащим параметром которой является полученный ранее идентификатор криптографического контекста.

Работа с ключами с помощью CryptoAPI

Для генерации ключей в CryptoAPI предусмотрены две функции – **CryptGenKey** и **CryptDeriveKey**. Первая из них генерирует ключи случайным образом и используется при создании случайных сессионных ключей и ключей асимметричных криптоалгоритмов.

```
DWORD    dwFlags;
HCRYPTKEY hKey;

dwFlags = CRYPT_EXPORTABLE;
if (!CryptGenKey(hProvider, Algid, dwFlags, &hKey))
  return 0;
```

Параметр Algid задает идентификатор алгоритма шифрования, для которого генерируется ключ (например, **CALG_3DES** для алгоритма 3-DES). При генерации ключевых пар RSA для шифрования и подписи используются специальные значения **AT_KEYEXCHANGE** и **AT_SIGNATURE**. Параметр dwFlags

задает различные опции ключа, которые зависят от алгоритма и провайдера. Старшие 16 битов этого параметра задают размер ключа для алгоритма RSA. Флаг **CRYPT_EXPORTABLE** позволяет экспортировать ключи симметричных и асимметричных алгоритмов из контейнера (по умолчанию это запрещено). Параметры, которые нельзя задать при генерации ключа (например инициализационные векторы), можно установить уже после его создания с помощью функции **CryptSetKeyParam**.

Функция **CryptDeriveKey** генерирует ключи на основе пользовательских данных. При этом гарантируется, что для одних и тех же входных данных **CryptDeriveKey** всегда выдает один и тот же результат. Этот способ генерации ключей может быть полезен для создания симметричного ключа шифрования на базе пароля. Работа с функцией **CryptDeriveKey** более трудоемка, поскольку она способна создавать ключ только на основе хеш-значения фиксированного размера, получаемого на основе пользовательских данных. Следует иметь в виду, что использованный в **CryptDeriveKey** хеш необходимо сразу же уничтожить и не использовать в дальнейших операциях.

```
HCRYPTKEY  hKey;
HCRYPTHASH hHash;

if (CryptCreateHash(hProv, CALG_MD5, 0, 0, &hHash))
{
  if (CryptHashData(hHash,(PBYTE)pPassword,strlen(pPassword),0))
  {
    if (!CryptDeriveKey(hProv,ENCRYPT_ALGORITHM,hHash,0,&hKey))
    {
      hKey = (HCRYPTKEY)NULL;
    }
  }
}
CryptDestroyHash(hHash);
```

Параметры ключей, которые не могут быть заданы при их создании, задаются с помощью функции **CryptSetKeyParam**. Например, использование блочного алгоритма в режиме обратной связи по шифротексту (CFB) задается в параметре KP_MODE. Для этого режима также можно выполнить инициализацию симметричного алгоритма стартовым вектором с помощью параметра KP_IV.

```
DWORD dwMode = CRYPT_MODE_CFB;
if(!CryptSetKeyParam(hKey, KP_MODE, (BYTE *)&dwMode, 0))
    return 0;
```

После окончания работы с ключом его нужно уничтожить вызовом **CryptDestroyKey**. При этом закрытый ключ сохраняется в контейнере (если, конечно, не использовался режим CRYPT_VERIFYCONTEXT), а сессионные ключи уничтожаются совсем.

Для передачи открытых ключей асимметричных алгоритмов между пользователями системы используются функции **CryptExportKey** и **CryptImportKey**. В качестве ключей экспорта/импорта могут использоваться либо ключевая пара RSA (с типом AT_KEYEXCHANGE), либо симметричный сеансовый ключ.

```
DWORD dwBlobLen;
BYTE *pbKeyBlob;

if(CryptExportKey(hKey,NULL,PUBLICKEYBLOB,0,NULL,&dwBlobLen))
{
  if((pbKeyBlob = (BYTE*)malloc(dwBlobLen)))
  {
    if(!CryptExportKey(hKey, NULL, dwBlobType,
       0, pbKeyBlob, &dwBlobLen))
      pbKeyBlob = (BYTE *)NULL;
  }
}
```

Параметр dwBlobType зависит от того, какой ключ экспортируется (импортируется), и задает тип структуры, в которую помещается экспортируемый ключ. Для открытого ключа это PUBLICKEYBLOB, и ключ экспорта/импорта при этом лишен смысла и должен быть нулем. Для закрытого ключа это PRIVATEKEYBLOB, и в качестве ключа экспорта может использоваться сеансовый ключ. Для сеансового ключа это обычно SIMPLEBLOB, а экспортируется он, как правило, на открытом ключе получателя. Параметры pbData и pdwDataLen задают адрес и размер буфера под структуру экспортируемого ключа. Если установить pbData в ноль, то функция вернет требуемый для размещения экспортируемого ключа размер в pdwDataLen.

Симметричное шифрование с помощью CryptoAPI

Для симметричного шифрования информации Microsoft CryptoAPI представляет целый ряд алгоритмов, реализуемых стандартными криптопровайдерами (табл. 1). Размер ключа определяет криптографическую стойкость алгоритма. На сегодняшний день длина ключей в алгоритмах, входящих в состав стандартного криптопровайдера MS_DEF_PROV, считается недостаточной. Последний разрабатывался в то время, когда в США действовал запрет на экспорт стойких криптоалгоритмов. Пришедший ему на смену криптопровайдер MS_ENHANCED_PROV устранил это ограничение. Использование CryptoAPI позволяет легко перейти с одного криптопровайдера на другой – с помощью замены всего одного параметра при вызове функции CryptAcquireContext.

Для шифрования/дешифрования данных, независимо от выбранного алгоритма, в CryptoAPI используются функции **CryptEncrypt** и **CryptDecrypt**. Эти функции можно использовать для шифрования как одного блока информации, так и больших объемов данных. Следует иметь в виду, что при использовании блочных шифров размер зашифрованного сообщения может быть больше размера исходного текста, вплоть до длины одного шифроблока.

Таблица 1

Симметричные алгоритмы шифрования, реализованные в CryptoAPI

Алгоритм	Провайдер	Идентификатор ALG_ID	Длина ключа, бит	Размер блока, бит
RC2	Base, Enhanced, AES	CALG_RC2	40	64
RC4	Base	CALG_RC4	40	–
RC4	Enhanced, AES	CALG_RC4	128	–
DES	Enhanced, AES	CALG_DES	56	64
2-key 3-DES	Enhanced, AES	CALG_3DES_112	112	64
3-key 3-DES	Enhanced, AES	CALG_3DES	168	64
AES	AES	CALG_AES_128	128	128
AES	AES	CALG_AES_192	192	128
AES	AES	CALG_AES_256	256	128

Рассмотрим пример использования функций шифрования CryptoAPI:

```
BYTE *pbData = NULL;
DWORD dwDataOutLen, dwDataInLen;

//размер блока исходного текста кратен размеру блока алгоритма
dwDataInLen = 1000 - 1000 % ENCRYPT_BLOCK_SIZE;

//размер блока шифротекста учитывает размер заполнителя
dwDataOutLen = dwDataInLen + ENCRYPT_BLOCK_SIZE;

if ((pbData = new BYTE [dwDataOutLen + 1]) == NULL)
  return 0;

do {
  dwCount = fread(pbData, 1, dwDataOutLen, *hSource);
  if (ferror(*hSource)) break;
  bFinal = (feof(*hSource) ? TRUE : FALSE);
  if (!CryptEncrypt(hKey,hHash,bFinal,0,
    pbData,&dwDataInLen,dwDataOutLen))
    break;
  fwrite(pbData, 1, dwDataInLen, *hDestination);
  if(ferror(*hDestination)) break;
}
while (!feof(*hSource));

if(pbData) delete []pbData;
```

В приведенном примере выполняется шифрование файла, открытого с помощью идентификатора **hSource**, результат помещается в файл **hDestination**. Параметр **hHash** позволяет параллельно с шифрованием/дешифрованием проводить хеширование данных для последующей электронной подписи или ее про-

верки. Флаг **bFinal** определяет, является ли шифруемый блок данных последним. Он необходим, поскольку для последнего блока всегда выполняется определенная деинициализация алгоритма (освобождаются внутренние структуры) и многие алгоритмы производят добавление (и проверку корректности при дешифровании) заполнителя после основных данных. Параметры **pbData** и **dwDataLen** задают адрес буфера и размер шифруемых данных. Для непоследнего блока данных (**Final** = FALSE) размер данных должен быть всегда кратен размеру шифруемого алгоритмом блока (для 3DES и DES этот размер равен 64 битам). Для последнего блока допускается нарушение этого условия. Заметим, что зашифрованные данные помещаются в тот же самый буфер поверх исходных.

Задания

Для реализации всех заданий используйте криптографические возможности, предоставляемые интерфейсом Microsoft CryptoAPI. Варианты заданий представлены в табл. 2.

1. Реализуйте программное средство защищенного обмена файлами. В функции программы входят шифрование и дешифрование файлов на дисках с помощью алгоритма симметричного шифрования ALG. На вход программы подаются тип криптографической операции (шифрование/дешифрование), имя входного и выходного файлов и пароль. При выполнении шифрования на выходе получается зашифрованный файл, при выполнении дешифрования – файл, идентичный исходному (если введен правильный пароль).

2. Модифицируйте программу из задания 1 так, чтобы она позволяла выбирать для любого из доступных в системе симметричных алгоритмов шифрования.

3. Модифицируйте программу из задания 1 так, чтобы на ее входе не требовался пароль. Для шифрования файла используйте сессионный ключ, выбранный случайным образом. Для безопасного хранения этого ключа вместе с зашифрованным файлом используйте асимметричное шифрование с помощью пары ключей текущего пользователя системы, получаемой с помощью функции ***CryptGetUserKey***.

Таблица 2

Варианты заданий для работы № 1

№ варианта	1	2	3	4
ALG	CALG_RC2	CALG_RC4	CALG_DES	CALG_3DES_112
№ варианта	5	6	7	8
ALG	CALG_3DES	CALG_AES_128	CALG_AES_192	CALG_AES_256

2. АУТЕНТИФИКАЦИЯ СООБЩЕНИЙ

Шифрование защищает от атак пассивной формы (подслушивания). Совершенно иной проблемой является защита от активных атак (фальсификации данных и транзакций). Защита от таких атак обеспечивается аутентификацией сообщений. Говорят, что сообщение, файл, документ или какой-то другой набор данных является ***аутентичным*** (т.е. подлинным), если такой набор данных действительно получен из того источника, который объявлен в сообщении, и в точности соответствует тому набору данных, которые из этого источника отсылались. ***Аутентификация сообщений*** представляет собой процедуру, обеспечивающую связывающимся сторонам возможность проверки аутентичности получаемых сообщений.

Большинство алгоритмов аутентификации сообщений можно реализовать при помощи криптографически стойких функций хеширования. Функция H называется криптографически стойкой, если она обладает следующими свойствами:

1. Применима к блоку данных любой длины.
2. Дает на выходе значение фиксированной длины.
3. Значение $H(x)$ вычисляется относительно легко для любого заданного x.
4. Чувствительна ко всевозможным изменениям в тексте x, таким как вставки, выбросы, перестановки и т.п.
5. Для любого данного кода h должно быть практически невозможно вычислить x, для которого $H(x) = h$ (свойство необратимости).
6. Для любого блока x должно быть практически невозможно вычислить $y \neq x$, для которого $H(x) = H(y)$.
7. Должно быть практически невозможно вычислить любую пару различных значений x и y, для которых $H(x) = H(y)$.

Вычисленное и переданное с сообщением хеш-значение может быть использовано для проверки его корректности в качестве контрольной суммы, но для целей аутентификации оно должно быть дополнительно защищено. Рассмотрим возможные схемы аутентификации на основе хеш-функций.

1. Шифрование хеш-значения симметричным методом

Для проверки корректности сообщения в этом случае используется код аутентификации MAC (Message Authentication Code), который представляет собой зашифрованное значение хеш-функции от сообщения M (рис. 2). Злоумышленник не может изменить сообщение без отражения этого факта на значении MAC. Получатель может быть уверен, что сообщение пришло из указанного источника, поскольку секретный ключ никому, кроме указанного отправителя, не известен.

2. Хеширование с использованием секретного значения

Этот вариант аутентификации сообщения с помощью функции хеширования не использует шифрования вообще (рис. 3). Сообщающиеся стороны, скажем, A и B, имеют известное только им общее секретное значение S_{AB}. Перед отсылкой сообщения стороне B сторона A вычисляет функцию хеширования для результата конкатенации этого секретного значения и текста сообще-

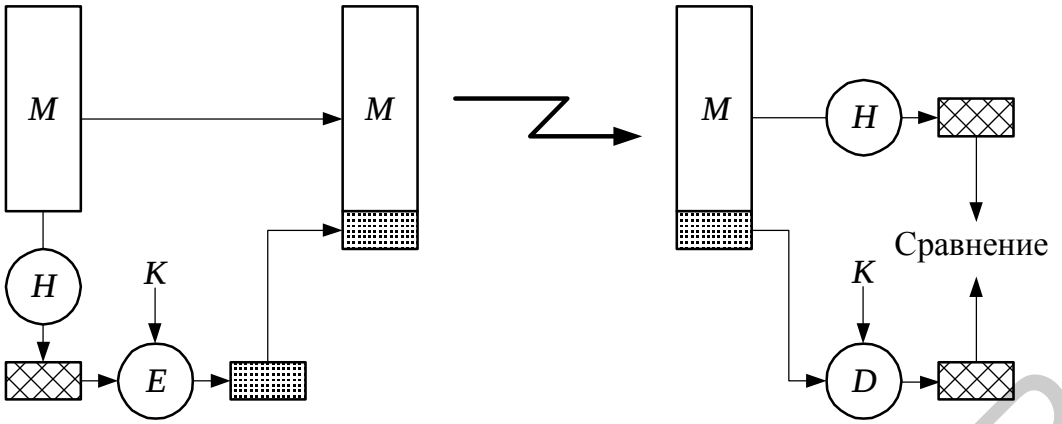

Рис. 2. Аутентификация симметричным шифрованием хеш-значения

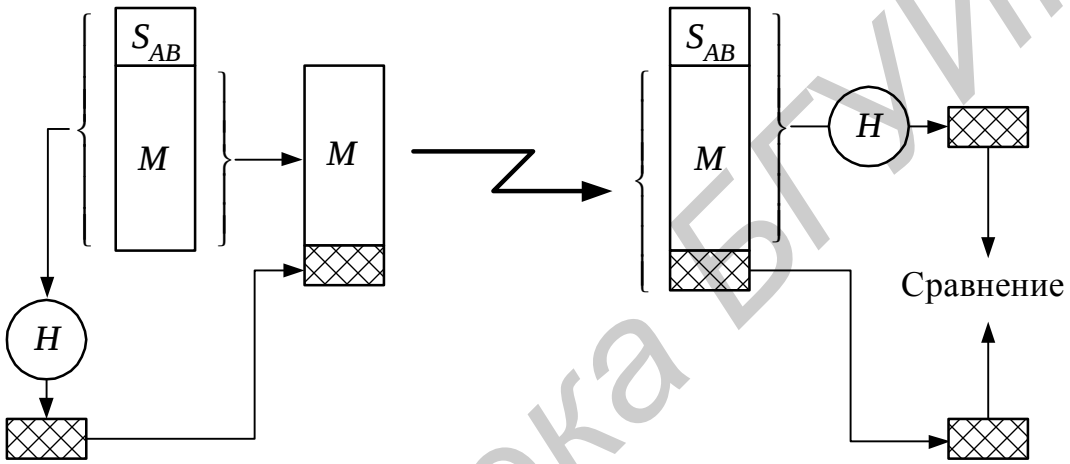

Рис. 3. Аутентификация хешированием секретного значения

ния: $D_M = H(S_{AB}||M)$. Затем $[M||D_M]$ пересылается стороне B. Поскольку сторона B имеет значение S_{AB}, она может вычислить $H(S_{AB}||M)$ и проверить соответствие вычисленного значения полученному значению D_M. Поскольку само секретное значение с сообщением не пересылается, у нарушителя нет возможности модифицировать перехваченное сообщение. Пока секретное значение остается секретным, нарушитель не может генерировать фальшивые сообщения. При этом алгоритм аутентификации выполняется за время, соизмеримое со временем выполнения функции хеширования.

Один из вариантов реализации данного подхода, называемый HMAC, принят в качестве стандарта защиты протоколов IPSec, TLS, SET и др. Его работа описывается выражением:

$$\text{HMACK} = H[(K+ \oplus \text{opad}) \,||\, H[(K+ \oplus \text{ipad}) \,||\, M]],$$

где H – встроенная функция хеширования (например SHA1);

M – подаваемое на вход HMAC сообщение (включая биты заполнителя, требуемые встроенной функцией хеширования);

K – секретный ключ;

$K+$ – ключ K, расширенный до размера хеш-значения путем добавления нулей слева;

ipad – шестнадцатеричное 363636…36, повторенное до заполнения $K+$;

opad – шестнадцатеричное 5C5C5C…5C, повторенное до заполнения *K*+.

Связывание с ipad означает переключение половины битов *K*. Точно так же связывание с opad означает переключение половины битов *K*, но для другого набора битов. В результате этого из *K*+ получаются два ключа, сгенерированных псевдослучайным образом, что повышает криптостойкость алгоритма.

3. Шифрование хеш-значения асимметричным методом

Хеш-значение сообщения передается в зашифрованном закрытым ключом отправителя K_S виде, составляя цифровую подпись сообщения (рис. 4). Получатель может проверить правильность цифровой подписи, используя открытый ключ K_O отправителя для дешифрования хеш-значения. Это доказывает, что тот, кто указан в качестве отправителя сообщения, является его создателем и что сообщение не было впоследствии изменено другим человеком, так как только отправитель владеет своим закрытым ключом, использованным для формирования цифровой подписи.

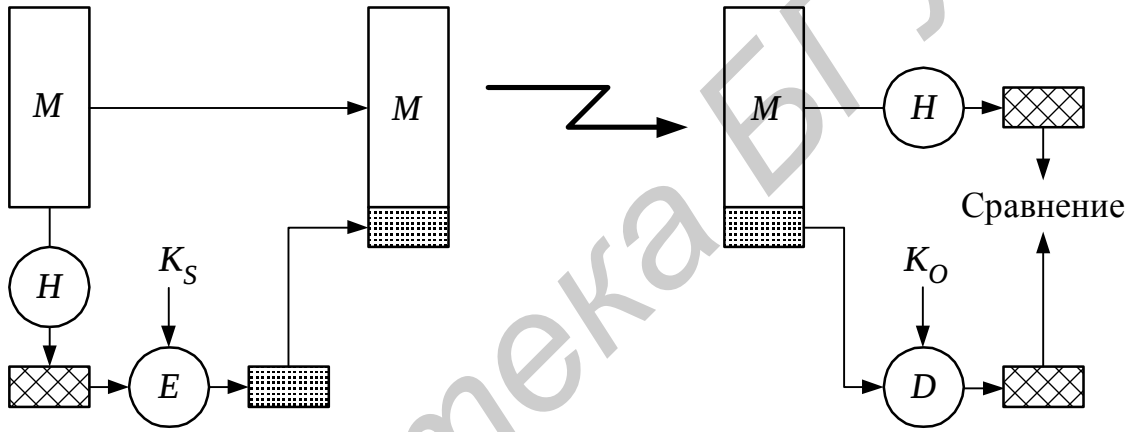

Рис. 4. Аутентификация асимметричным шифрованием хеш-значения

Такой подход имеет два преимущества: он кроме аутентификации сообщений обеспечивает также цифровую подпись и не требует доставки секретного ключа сообщающимся сторонам. Кроме того, он защищает от «ренегатства» – отказа лица, поставившего подпись, от нее в последующем.

Вычисление хешей с помощью CryptoAPI

Хеш создается вызовом функции **CryptCreateHash**, принимающей на входе контекст криптопровайдера, идентификатор алгоритма (**CALG_MD5** или **CALG_SHA**) и дескриптор ключа (для автоматического вычисления кодов аутентификации MAC и HMAC). После этого хеш можно вычислять как явно, используя функцию **CryptHashData**, так и неявно, передавая дескриптор хеша в функцию **CryptEncrypt**. Вычисление хеша (так же, как и шифрование) можно производить по частям (например, читая данные в буфер и хешируя содержимое буфера в цикле). После окончания хеширования можно либо подписать хеш, либо получить его значение вызовом функции CryptGetHashParam с параметром HP_HASHVAL. После получения значения хеш использовать уже нельзя. Его нужно разрушить вызовом **CryptDestroyHash**. Проверка хеша произво-

дится точно так же, как и его создание, – нужно вычислить хеш и сверить полученное значение с сохраненным:

```
HCRYPTHASH hHash;
CryptCreateHash(hProv, CALG_MD5, 0, 0, &hHash);
CryptHashData(hHash,(BYTE *)&data, dwLen, 0);

BYTE newHash[16];
dwLen = 16;
CryptGetHashParam(hHash, HP_HASHVAL, newHash, &dwLen, 0);
if(!memcmp(newHash, oldHash, 16))
{
    //хеш верен
}
else
{
    //хеш не верен
}
CryptDestroyHash(hHash);
```

Электронная цифровая подпись с помощью CryptoAPI

Функции CryptoAPI позволяют подписывать только хеш-значения, но не сами данные. Хеш предварительно должен быть вычислен, а затем подписан с помощью функции **CryptSignHash** на закрытом ключе (AT_KEYEXCHANGE), находящемся в контейнере криптопровайдера. Проверка подписи осуществляется на открытом ключе AT_SIGNATURE, который нужно предварительно импортировать:

```
CryptSignHash(hHash, AT_SIGNATURE, NULL, 0, buf, &dwLen);
if(CryptVerifySignature(hHash, buf, 1024/8, hPubKey, NULL, 0))
{
  // Верная подпись
}
else
{
  // Неверная подпись или хеш
}
```

Размер подписи равняется размеру ключа асимметричного алгоритма, использованного для ее постановки (RSA или DSS). Вычисление/проверка подписи RSA и DSS выполняется одинаково, однако для работы с подписью DSS нужно использовать Microsoft DSS Cryptographic Provider (**MS_DEF_DSS_PROV**, тип **PROV_DSS**). Кроме того, подпись DSS работает только с алгоритмом хеширования SHA.

Задания

1. Реализуйте аутентификацию сообщений на основе алгоритма HMAC средствами CryptoAPI двумя способами:

а) используя криптоинтерфейс только для хеширования значений (конкатенацию сообщения и секретного значения выполнять самостоятельно);

б) используя вызов функции CryptCreateHash с ключевым значением для вычисления кода аутентификации HMAC средствами криптоинтерфейса.

2. Реализуйте программное средство постановки электронной цифровой подписи для дисковых файлов с помощью соответствующего криптоалгоритма ALG1. На вход системы при постановке подписи подаются имя подписываемого файла и имя файла цифровой подписи, при проверке подписи входными данными являются имя подписанного файла и файла подписи, а также имя пользователя, подлинность подписи которого проверяется. В качестве алгоритма хеширования используйте криптоалгоритм ALG2.

Таблица 3

Варианты заданий для работы № 2

№ варианта	Алгоритм	
	ALG1	ALG2
1	CALG_RSA_SIGN	CALG_MD2
2	CALG_RSA_SIGN	CALG_SHA
3	CALG_RSA_SIGN	CALG_MD5
4	CALG_RSA_SIGN	CALG_SHA
5	CALG_DSS_SIGN	CALG_SHA
6	CALG_RSA_SIGN	CALG_MD2
7	CALG_RSA_SIGN	CALG_SHA
8	CALG_RSA_SIGN	CALG_MD5
9	CALG_RSA_SIGN	CALG_SHA

3. ИДЕНТИФИКАЦИЯ И АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ

Идентификацию и аутентификацию пользователей можно считать основой программно-технических средств безопасности, поскольку остальные сервисы рассчитаны на обслуживание именованных субъектов. ***Идентификация*** позволяет субъекту (пользователю, процессу, действующему от имени определенного пользователя, или иному аппаратно-программному компоненту) назвать себя (сообщить свое имя). Посредством ***аутентификации*** вторая сторона убеждается, что субъект – действительно тот, за кого он себя выдает. Совокупность выполнения процедур идентификации и аутентификации называют процедурой ***авторизации***.

Основные способы аутентификации пользователей представлены на рис. 5. При прямой аутентификации (рис. 5, *а*) сервер опознает пользователя по наличию общей секретной информации, передаваемой вместе с запросом. Такая схема эффективно работает при локальной аутентификации и реализуется в процедурах входа большинства операционных систем. Однако для удаленной аутентификации она в явном виде неприменима. Злоумышленник, перехватив секретные данные, пусть даже в зашифрованном виде, может использовать их для дальнейшего самостоятельного доступа к ресурсам сервера (атака воспроизведения). Для решения этой проблемы используются одноразовые пароли и протоколы идентификации с нулевым знанием.

Современные средства идентификации/аутентификации должны поддерживать ***концепцию единого входа в сеть***. Если в корпоративной сети много информационных сервисов, допускающих независимое обращение, то многократная идентификация/аутентификация становится слишком обременительной для пользователя, а настройка прав доступа на каждой машине затрудняет работу администраторов. Для решения этой проблемы используют методы аутентификации с помощью доверенного посредника, хранящего все учетные записи пользователей в централизованном хранилище безопасности. Пользователь сначала обращается к посреднику за разрешением на доступ к серверу ресурсов (рис. 5, *б*). Полученное разрешение может передаваться и клиенту, и серверу, или только клиенту (при этом пользователь доставляет разрешение на сервер самостоятельно). Возможно расширение этой схемы, когда требуется дополнительная аутентификация для доступа к посреднику, как и к любому ресурсному серверу (рис. 5, *в*).

Протокол S/KEY

Наиболее известным программным генератором ***одноразовых паролей*** является система S/KEY компании Bellcore (Internet-стандарт RFC 1938). Пусть имеется односторонняя хеш-функция f. Эта функция известна и пользователю, и серверу аутентификации. Пусть, далее, имеется секретный ключ K, известный только пользователю. На этапе начального администрирования пользователя функция f применяется к ключу K n раз, после чего результат сохраняется на

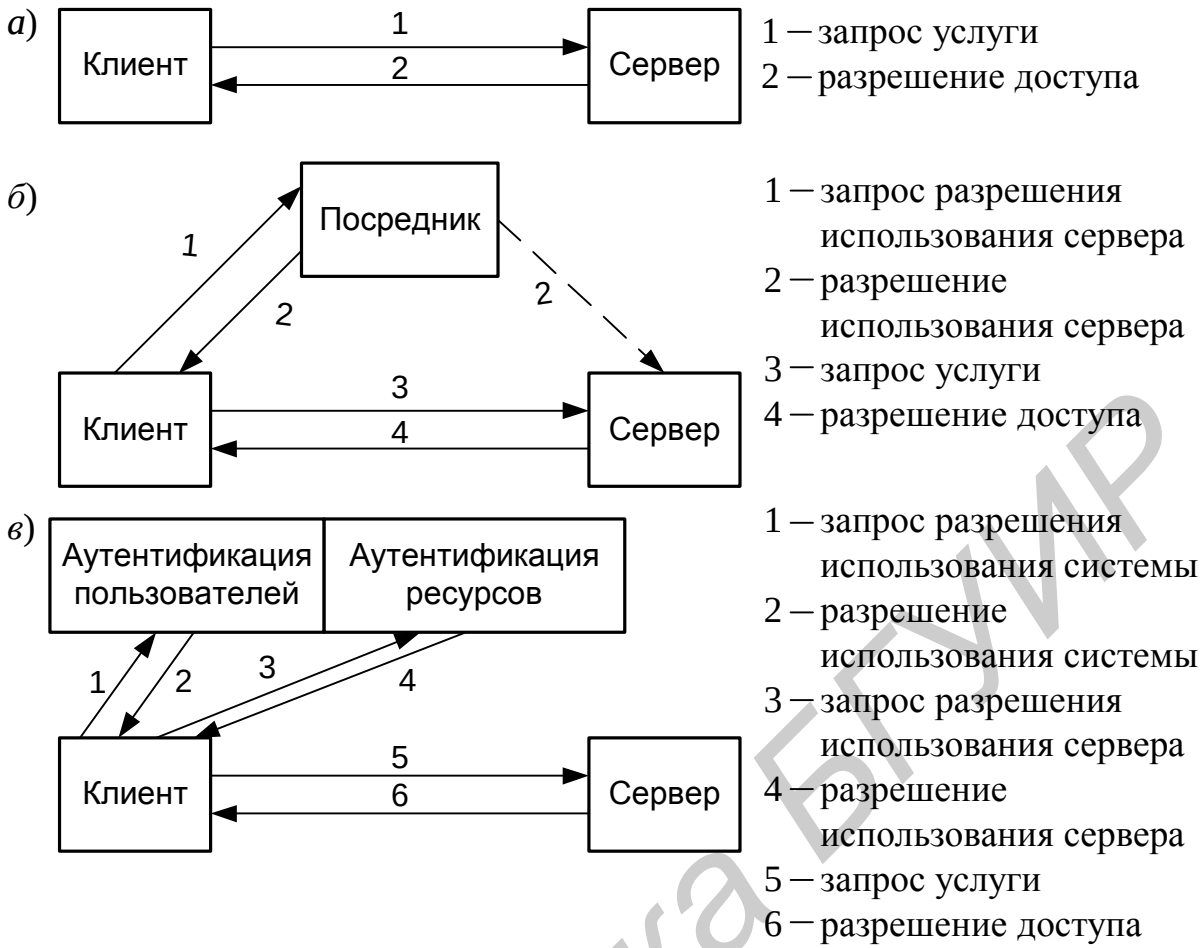

Рис. 5. Схемы аутентификации пользователей: *а*) – прямая; *б*) – с доверенным посредником; *в*) – с аутентификацией на доверенном посреднике

сервере. После этого процедура проверки подлинности пользователя выглядит следующим образом:

1. Сервер присылает на пользовательскую систему число $(n - 1)$.
2. Пользователь применяет функцию f к секретному ключу K $(n - 1)$ раз и отправляет результат по сети на сервер аутентификации.
3. Сервер применяет функцию f к полученному от пользователя значению и сравнивает результат с ранее сохраненной величиной.

В случае совпадения подлинность пользователя считается установленной, сервер запоминает новое значение (присланное пользователем) и уменьшает на единицу счетчик (n). Поскольку функция f необратима, то перехват пароля, равно как и получение доступа к серверу аутентификации, не позволяет узнать секретный ключ K и предсказать следующий одноразовый пароль.

Протокол Kerberos

Протокол Kerberos был создан в Массачусетском технологическом институте в рамках проекта Athena в конце 1980-х гг. Протокол Kerberos использует дополнительную аутентификацию на доверенном посреднике (рис. 5, *в*). Роль посредника здесь играет так называемый ***центр распределения ключей KDC*** (Key Distribution Center).

KDC представляет собой службу, работающую на физически защищенном сервере. Она ведет базу данных с информацией об учетных записях всех своих абонентов безопасности. Вместе с информацией о каждом из них в базе данных KDC сохраняется криптографический ключ, известный только этому абоненту и службе KDC. Данный ключ, который называют ***долговременным***, используется для связи пользователя системы безопасности с центром распределения ключей. В большинстве практических реализаций протокола Kerberos долговременные ключи создаются на основе пароля пользователя.

Путь клиента к ресурсу в системе Kerberos состоит из трех этапов:

1. Определение легальности клиента, осуществляющего вход в систему (сообщения 1 и 2 на рис. 5, *в*).
2. Получение разрешения на обращение к ресурсному серверу (сообщения 3 и 4).
3. Получение доступа к ресурсу (сообщения 5 и 6).

Для решения первой и второй задач клиент обращается к серверу KDC, который соответственно делится на две вспомогательные службы. Выполнение первичной аутентификации осуществляется так называемым ***аутентификационным сервером AS*** (Authentication Server). Этот сервер хранит в своей базе данных информацию об идентификаторах и паролях пользователей. Вторую задачу, связанную с получением разрешения на обращение к ресурсному серверу, решает ***сервер мандатов TGS*** (Ticket-Granting Server). Сервер TGS для легальных клиентов выполняет дополнительную проверку и дает клиенту разрешение на доступ к нужному ему ресурсному серверу, для чего наделяет его электронной формой-мандатом. Для выполнения своих функций сервер мандатов использует копии секретных ключей всех ресурсных серверов, которые хранятся у него в базе данных. Кроме этих ключей, сервер TGS имеет еще один секретный DES-ключ, который разделяется с сервером AS.

Протокол Kerberos описывается следующими сообщениями:

1. Клиент-AS: C, TGS.
2. AS-клиент: $\{K_{C,TGS}\}K_C$, $\{T_{C,TGS}\}K_{TGS}$.
3. Клиент-TGS: S, $\{A_{C,\,TGS}\}K_{C,TGS}$, $\{T_{C,TGS}\}K_{TGS}$.
4. TGS-клиент: $\{K_{C,S}\}K_{C,TGS}$, $\{T_{C,S}\}K_S$.
5. Клиент-сервер: $\{A_{C,S}\}K_{C,S}$, $\{T_{C,S}\}K_S$.
6. Сервер-клиент: $\{t + 1\}K_{C,S}$.

Здесь C обозначает клиента, S – сервер, K_x – секретный ключ x, $K_{x,y}$ – сеансовый ключ для x и y, $\{m\}K_x$ – сообщение m, шифрованное секретным ключом x, $T_{x,y}$ – мандат (билет) x на использование y, $A_{x,y}$ – удостоверение (аутентификатор) x для y, t – метка времени, входящая в состав удостоверения.

Мандат используется для безопасной передачи серверу личности клиента, которому выдан этот мандат. Мандат Kerberos имеет следующую форму:

$T_{C,S} = S, \{C, a, v, K_{C,S}\}K_S$.

Мандат пригоден для одного сервера и одного клиента. Он содержит имя клиента C, его сетевой адрес a, имя сервера S, срок действия v и сеансовый ключ

$K_{C,S}$. Эта информация шифруется секретным ключом сервера K_S. Если клиент получил мандат, он может использовать его для доступа к серверу много раз – пока не истечет срок действия мандата. Клиент не может расшифровать мандат (он не знает секретного ключа сервера), но он может предъявить его серверу в зашифрованной форме. Прочитать или изменить мандат при передаче его по сети невозможно.

Удостоверение – это дополнительный атрибут, предъявляемый вместе с мандатом, который преследует две цели. Во-первых, удостоверение содержит некоторый открытый текст, зашифрованный сеансовым ключом. Это доказывает, что клиенту известен ключ. Кроме того, зашифрованный открытый текст включает метку времени. Злоумышленник, которому удалось записать и мандат, и удостоверение, не сможет использовать их спустя два дня. Удостоверение Kerberos имеет следующую форму:

$A_{C,S} = \{C, t, ключ\}K_{C,S}$.

Клиент создает его каждый раз, когда ему нужно воспользоваться услугами сервера. Удостоверение содержит имя клиента C, метку времени t и необязательный дополнительный сеансовый ключ. Все эти данные шифруются сеансовым ключом $K_{C,S}$, общим для клиента и сервера. В отличие от мандата, удостоверение используется только один раз. Однако это не проблема, так как клиент может генерировать удостоверения по мере надобности (ему известен общий секретный ключ).

Этапы аутентификации по протоколу Kerberos:

1. Получение первоначального мандата. У клиента есть часть информации, доказывающей его личность, – его пароль. Однако он не пересылается по сети. Клиент посылает в адрес сервера аутентификации сообщение 1, содержащее только его имя и имя его сервера TGS. В случае обнаружения данных о пользователе в базе данных сервер AS генерирует сеансовый ключ, который будет использоваться для обмена данными между клиентом и TGS, и шифрует этот сеансовый ключ секретным ключом клиента. Затем он создает мандат для доступа клиента к серверу TGS (***мандат на выделение мандата*** TGT, Ticket Granting Ticket) и шифрует его секретным ключом TGS. В сообщение 2 включаются зашифрованный сеансовый ключ и мандат. Легитимный клиент может расшифровывать только первую часть сообщения. Злоумышленник не может расшифровать ни одну часть сообщения 2.

2. Получение серверных мандатов. Мандаты для получения доступа к конкретным серверам выдает центр распределения мандатов TGS при поступлении запросов от клиентов с корректными TGT и удостоверениями (сообщение 3). Сервер TGS, получив запрос, расшифровывает TGT своим секретным ключом. Затем TGS использует включенный в TGT сеансовый ключ, чтобы расшифровать удостоверение. Наконец, TGS сравнивает информацию удостоверения с информацией мандата, сетевой адрес клиента – с адресом отправителя запроса и метку времени – с текущим временем. Если все совпадает, TGS разрешает выполнение запроса. Проверка меток времени предполагает, что ча-

сы всех компьютеров синхронизированы с точностью, по крайней мере, до нескольких минут.

В ответ на правильный запрос TGS возвращает правильный мандат, который клиент может предъявить серверу. TGS также создает новый сеансовый ключ для клиента и сервера, зашифрованный сеансовым ключом, общим для клиента и TGS. Оба этих значения отправляются клиенту (сообщение 4). Клиент расшифровывает сообщение и извлекает сеансовый ключ.

3. Запрос услуги. Теперь клиент может доказать свою подлинность серверу. Клиент создает удостоверение и вместе с полученным в TGS мандатом передает запрос на сервер ресурсов (сообщение 5). Сервер расшифровывает и проверяет мандат и удостоверение, а также проверяет адрес клиента и метку времени. Если все в порядке, то сервер уверен, что, согласно Kerberos, клиент – именно тот, за кого он себя выдает. Если приложение требует взаимной проверки подлинности, сервер посылает клиенту сообщение, состоящее из метки времени, зашифрованной сеансовым ключом (сообщение 6). Это доказывает, что серверу известен правильный секретный ключ и он может расшифровать мандат и удостоверение. При необходимости клиент и сервер могут шифровать дальнейшие сообщения общим ключом. Так как этот ключ известен только им, они оба могут быть уверены, что последнее сообщение, зашифрованное этим ключом, отправлено другой стороной.

Если система Kerberos реализована и сконфигурирована правильно, она незначительно уменьшает производительность сети. Так как мандаты используются многократно, то сетевые ресурсы, затрачиваемые на запросы предоставления мандатов, невелики. Среди уязвимых мест системы Kerberos можно назвать централизованное хранение всех секретных ключей системы. Успешная атака на Kerberos-сервер, в котором сосредоточена вся информация, критическая для системы безопасности, приводит к крушению информационной защиты всей сети.

Задания

1. Реализуйте средствами криптографического интерфейса CryptoAPI систему аутентификации на основе одноразовых паролей по алгоритму S/KEY.

2. Реализуйте средствами криптографического интерфейса CryptoAPI систему аутентификации на основе протокола Kerberos. Программа должна отображать все сообщения, передаваемые между субъектами взаимодействия в исходном и зашифрованном виде. Результатом успешной аутентификации является наличие общего сеансового ключа, сгенерированного сервером выдачи мандатов, у клиента и сервера ресурсов.

3. Предложите и реализуйте протокол междоменной аутентификации с помощью протокола Kerberos на основе билетов переадресации, которые представляют собой билеты TGT, зашифрованные с помощью междоменного ключа.

4. КОНТРОЛЬ ДОСТУПА

Цель управления доступом – это ограничение операций, которые может проводить зарегистрировавшийся в системе легитимный пользователь. Управление доступом указывает, чтó конкретно пользователь имеет право делать в системе, а также какие операции разрешены для выполнения приложениями, выступающими от имени пользователя. Таким образом, управление доступом предназначено для предотвращения действий пользователя, которые могут нанести вред системе, например, нарушить ее безопасность.

Существуют три метода управления доступом к объектам в системе:

1. Дискреционный метод контроля доступа обеспечивает защиту персональных объектов в системе. Контроль является дискреционным в том смысле, что владелец объекта сам определяет тех, кто имеет доступ к объекту, а также вид их доступа.
2. Обязательный метод контроля доступа является обязательным в том смысле, что пользователи не могут изменять стратегию доступа в отношении объектов. Управление доступом осуществляется на основе классификации объектов и субъектов системы по уровням безопасности.
3. Ролевой метод контроля доступа основан на максимальном приближении логики работы системы к реальному разделению функций персонала в организации и управлении доступом на основе типов активности пользователей в системе – их ролей.

Наибольшую популярность в настоящее время получил дискреционный (избирательный) контроль. Этот метод управляет доступом субъектов к объектам, базируясь на идентификационной информации субъекта и списка доступа объекта, содержащего набор субъектов (или групп субъектов) и ассоциированных с ними типов доступа (например, чтение и запись). При запросе доступа к объекту система ищет субъекта в списке прав доступа объекта и разрешает доступ, если субъект присутствует в списке и разрешенный тип доступа включает требуемый тип. Иначе доступ не предоставляется. Очевидным примером использования дискреционного метода является система Windows NT/2k/XP.

Контроль доступа в ОС Windows NT

В основе защитной системы Windows NT лежат две составляющие – базы данных учетных записей пользователей и списки управления доступом (рис. 6).

База данных пользователей может содержаться на локальной машине (управляется диспетчером учетных записей безопасности SAM – Security Accounts Manager) или же реплицироваться из сетевой службы каталогов Active Directory. Каждая запись такой базы данных содержит сведения об имени пользователя, его пароле (хранится в зашифрованном виде) и его специальном уникальном идентификаторе безопасности SID (Security Identifier), используемом в качестве указателя на нее другими компонентами системы безопасности.

SID представляет собой числовое значение переменной длины, формируемое из номера версии структуры SID, 48-битного кода агента идентификатора и

переменного количества 32-битных кодов субагентов и/или относительных идентификаторов (relative identifiers, RID). Код агента идентификатора (identifier authority value) определяет агент, выдавший SID. Таким агентом обычно является локальная система или домен под управлением Windows. Коды субагентов идентифицируют попечителей, уполномоченных агентом, который выдал SID, а RID – не более чем средство создания уникальных SID на основе общего базового SID (common-based SID). Поскольку длина SID довольно велика и Windows старается генерировать истинно случайные значения для каждого SID, вероятность появления двух одинаковых SID практически равна нулю.

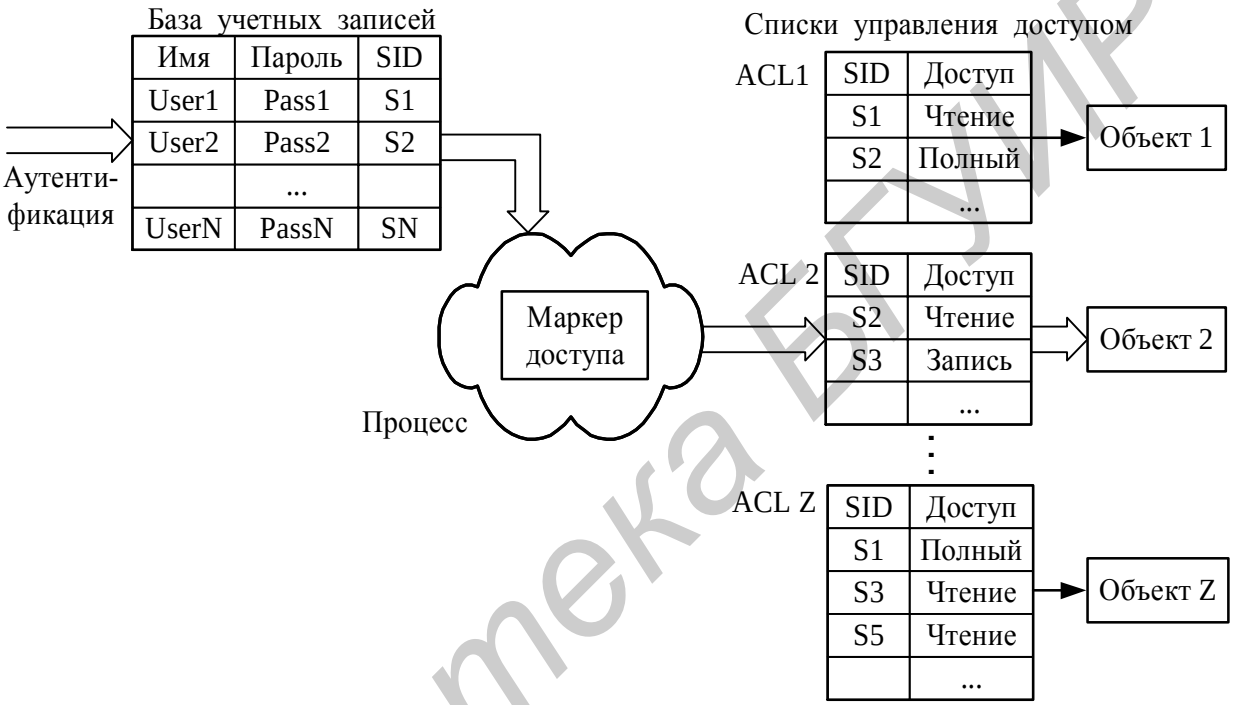

Рис. 6. Упрощенная схема работы системы безопасности Windows NT

Получить значение SID текущего пользователя можно на основе его имени с помощью функции **LookupAccountName**. С помощью этой функции можно получить SID любой учетной записи в системе. Для создания произвольного значения SID предназначена функция **AllocateAndInitializeSid.**

```
DWORD dwUserBuf = 256;
char  chCurrentUser[256] = {0};
GetUserName(chCurrentUser, &dwUserBuf);

PSID  userSID;
DWORD dwSID = 1024*512;
char  szRef[MAX_PATH]={0};
DWORD cbRef = sizeof(szRef);
LookupAccountName(NULL, chCurrentUser, &userSID,
  &dwSID, szRef, &cbRef, NULL);
```

Некоторым стандартным учетным записям и группам ОС Windows выдает SID, состоящие из SID компьютера или домена и предопределенного RID. Так, RID

учетной записи администратора равен 500, а RID гостевой учетной записи – 501. Для получения доступа к таким SID предназначена функция **CreateWellKnownSid**. В следующем примере создается SID для группы Everyone (Все):

```
DWORD SidSize;
PSID  hSID;
SidSize = SECURITY_MAX_SID_SIZE;
if(!(hSID = LocalAlloc(LMEM_FIXED, SidSize)))
  exit(1);
if(!CreateWellKnownSid(WinWorldSid, NULL, hSID, &SidSize))
  fprintf(stderr, "Ошибка CreateWellKnownSid %u", GetLastError());

// ...

LocalFree(hSID);
```

Значение SID пользователя и всех групп, в которые он входит, сохраняется после аутентификации в специальной структуре – маркере доступа – для каждого запускаемого пользователем процесса и проверяется при доступе к любому защищаемому объекту. Объектами защиты ОС Windows NT являются локальные файлы и каталоги, разделяемые файлы, разделы реестра, общая память, объекты-задания, мьютексы, семафоры, именованные каналы, принтеры, объекты каталога Active Directory и пр. С каждым защищаемым объектом ассоциируется дескриптор защиты (Security Descriptor), указывающий, кому и какие действия разрешено выполнять над данным объектом. Дескриптор защиты хранит:

– флаги, определяющие поведение или характеристики дескриптора. Например, флаг SE_DACL_PROTECTED запрещает наследование дескриптором параметров защиты от другого объекта;

– идентификатор защиты SID владельца;

– идентификатор защиты SID основной группы;

– список управления избирательным доступом DACL (discretionary access-control list) указывает, кто может получать доступ к объекту и вид доступа;

– системный список управления доступом SACL (system access-control list) указывает, какие операции и каких пользователей должны регистрироваться в журнале аудита безопасности.

Список DACL представляет собой таблицу управляющих записей ACE (Access-Control Entries), каждая из которых содержит SID учетной записи и маску доступа. ACE бывают либо запрещающими, либо разрешающими. При попытке пользовательского процесса получить доступ к защищенному объекту система безопасности сравнивает его идентификатор защиты и идентификаторы групп, в которые он входит, с записями ACL из списка управлением доступом по следующему алгоритму:

1. В отсутствие DACL (DACL = null) объект является незащищенным и система защиты предоставляет к нему запрошенный тип доступа.

2. Если у вызывающего потока имеется привилегия на захват объекта во владение, система защиты предоставляет ему право на доступ для записи в DACL.

3. Если вызывающий поток является владельцем объекта, ему предоставляются права управления чтением и записью в DACL.

4. Просматриваются все ACE в DACL – от первого к последнему. В случае совпадения SID из запрещающего ACE с SID из маркера доступа вызывающего потока доступ к объекту не предоставляется. В случае разрешительной записи ACE соответствующие ей права добавляются в метку разрешенных прав и просмотр ACL продолжается. Доступ предоставляется, если процессу предоставлены все запрошенные им права.

5. Если достигнут конец DACL и некоторые из запрошенных прав доступа еще не предоставлены, доступ к объекту запрещается.

Создание эффективного списка DACL является необходимым условием грамотного управления доступом к объекту. Для его создания формируется массив управляющих записей, из которых с помощью функции **SetEntriesInAcl** формируется список DACL или SACL, который затем заносится в дескриптор безопасности объекта с помощью функции **SetSecurityDescriptorDacl**. Просмотреть дескриптор безопасности открытого объекта можно с помощью функции **GetSecurityInfo**. Более удобным средством описания списков DACL в ОС Windows версии 2000 и последующих версий является язык SDDL.

Язык SDDL

SDDL (Security Descriptor Definition Language) – это язык, представляющий дескриптор безопасности в текстовой, понятной человеку форме. С помощью SDDL можно описать владельца объекта, основную группу, DACL и SACL. Синтаксис SDDL имеет следующую форму:

```
O:sid_владельца
G:sid_основной_группы
D:флаги_DACL(ACE 1)(ACE 2)…(ACE n)
S:флаги_SACL(ACE 1)(ACE 2)…(ACE n)
```

Поля SID в строках SDDL, описывающих владельца или основную группу, могут содержать либо SID учетной записи, либо одно из стандартных значений (табл. 4).

DACL- и SACL-флаги используют одни и те же значения и применяются для управления наследованием параметров. Например, флаг «P» запрещает наследование параметров от родителя, а флаг «AI» включает автоматическое наследование всех ACE данного объекта его потомками.

Записи ACE содержат всю информацию, описывающую права доступа и аудита для любой учетной записи системы. В языке SDDL они имеют следующий формат:

```
(тип_ACE;флаги_ACE;права;GUID_объекта;GUID_наследуемого_объекта;SID)
```

Записи ACE могут принадлежать к одному из основных типов – разрешительные (обозначаются «A») и запретительные (обозначаются «D»). Флаги ACE предназначены для тех же целей, что и флаги всей ACL, – управляют наследо-

Таблица 4

Стандартные SID, используемые в SDDL

Псевдоним	Описание
AN	Anonymous – анонимные пользователи
BA	Built-in Administrators – встроенная группа администраторов
BG	Built-in Guests – встроенная группа гостей
BU	Built-in Users – встроенная группа пользователей
CO	Creator Owner – создатель-владелец
WD	World (Everyone) – все пользователи
IU	Interactive Users – интерактивные пользователи
LA	Local Administrator – администратор локального компьютера
LG	Local Guest – гость локального компьютера
SY	Local System – локальная учетная запись системы
PU	Power Users – опытные пользователи
SU	Service Logon User – пользователи, вошедшие в систему от имени системной службы

ванием прав доступа. Например, флаг «CI» разрешает потомкам-контейнерам наследовать данный элемент ACE, флаг «OI» разрешает аналогичное действие для потомков-объектов.

Права доступа определяют действия, которые ассоциированы с данным элементом ACE. Некоторая часть действий применима ко всем типам объектов (чтение, запись, удаление), другая часть описывает специализированные действия (запуск дискового файла на выполнение, создание дочернего объекта Active Directory и пр.). Основные права, поддерживаемые всеми типами объектов, перечислены в табл. 5.

Таблица 5

Права доступа в ACE

Обозначение	Описание
GA	Полный доступ
GR	Чтение
GW	Запись
GX	Выполнение
RC	Чтение данных ACL
SD	Удаление объекта
WD	Изменение DACL
WO	Установка владельца

В поле GUID_объекта при необходимости (в случае работы с объектами Active Directory) подставляется глобальный идентификатор GUID соответствующего объекта. Поле GUID_наследуемого_объекта содержит GUID типа объекта, которому должны отвечать все объекты-потомки.

Рассмотрим пример SDDL- строки:

```
D:P(D;OICI;GA;;;S-1-5-21-220523388-854245398-1343024091-1004)
   (A;OICI;GA;;;BA)
   (A;OICI;GRGWGX;;;IU)
   (A;OICI;GR;;;BG)
```

В этой записи описывается список DACL («D:»). Флаг «P» запрещает наследование параметров от родителя. Первый элемент ACE запрещает пользователю с идентификатором SID S-1-5-21-220523388-854245398-1343024091-1004 полный доступ к объекту. Второй ACE предоставляет неограниченные полномочия встроенной группе администраторов. Третий ACE разрешает интерактивным пользователям чтение, запись и модификацию объекта. Четвертый ACE ограничивает встроенную группу гостей возможностью только чтения данных. Во всех ACE используются флаги «OI» и «CI», активирующие наследование дочерними объектами и контейнерами параметров, описываемых ACE. В следующем примере создается каталог «C:\MyDir» с рассмотренным списком доступа:

```
SECURITY_ATTRIBUTES sa;
sa.nLength = sizeof(SECURITY_ATTRIBUTES);
sa.bInheritHandle = FALSE;
char *szSD = "D:P"
  "(D;OICI;GA;;;S-1-5-21-220523388-854245398-1343024091-1004)"
  "(A;OICI;GA;;;BA)(A;OICI;GRGWGX;;;IU)(A;OICI;GR;;;BG)";

if (ConvertStringSecurityDescriptorToSecurityDescriptor(
        szSD, SDDL_REVISION_1,
        &(sa.lpSecurityDescriptor), NULL))
{
        if (!CreateDirectory("C:\\MyDir", &sa ))
                exit(1);
        LocalFree(sa.lpSecurityDescriptor);
}
```

Задания

1. Разработайте программное средство для поиска на диске и в реестре ОС Windows объектов с нулевым списком DACL и вывода их списка.

2. Разработайте программное средство проверки доступа пользователя к объектам файловой системы и реестра ОС Windows. На вход программы подаются имя пользователя, имя объекта и тип доступа. В имени объекта может указываться маска перечисления для нескольких объектов. На выходе выдается список объектов, к которым пользователь может получить указанный во входных параметрах тип доступа.

3. Напишите программное средство просмотра и изменения списка управления доступом для заданного объекта. Программа должна выводить список элементов ACE из списка ACL объекта и позволять добавлять и удалять разрешения.

5. АТАКИ НА ПЕРЕПОЛНЕНИЕ БУФЕРА

Переполнение буфера (buffer overflow) – одна из наиболее распространенных удаленных атак, позволяющих злоумышленнику получить частичный или полный контроль над атакуемым хостом. Первая атака с применением данной уязвимости использовалась в вирусе-черве Морриса в 1988 году. С тех пор их число увеличивается с каждым годом.

Реализация атаки требует от злоумышленника решения двух задач:

1. Подготовка кода, который будет выполняться в контексте привилегированной программы.
2. Изменение последовательности выполнения программы с передачей управления подготовленному коду.

Подготавливаемый код представляет собой исполняемые машинные инструкции соответствующего процессора и может передаваться в программу в качестве ее параметра или команды. В некоторых случаях передачи нужного кода в программу не требуется, если он уже присутствует в ней самой или в ее адресном пространстве и требуется лишь его параметризация.

Передача управления подготовленному коду выполняется злоумышленником посредством переполнения буфера, т.е. блока памяти, выделенного под входную переменную. Переполнение возникает при отсутствии проверки выхода данных за границы буфера. В результате этого искажается содержимое других переменных или служебных данных программы, которые входят в область переполнения буфера. Основные причины переполнения буфера – плохой стиль кодирования (особенно это касается языков C и C++), отсутствие защищенных и простых в использовании строковых функций и непонимание последствий тех или иных ошибок.

Переполнение стека

Переполнение буфера в виде переполнения стека возникает, когда буфер, выделенный в стеке, перезаписывается данными, объем которых превосходит его размер. Чаще всего объектом изменения становится адрес возврата из текущей функции, размещенный в стеке вместе с переменными. На рис. 7 представлена программа на языке C, демонстрирующая самый простой метод эксплуатации переполнения.

Это приложение по простоте сродни программе «Hello, World!». Оно начинается с того, что выводит адрес функций *bar*. Для этого используется параметр «%p» функции *printf*. В функции *foo* объявлен статический буфер, переполнение которого можно использовать для несанкционированного вызова функции *bar*. Функция *foo* содержит пару вызовов *printf*, которые используют побочные свойства функции с переменным числом аргументов, чтобы напечатать содержимое стека. Проблемы начинаются, когда функция *foo* слепо принимает вводимые пользователем данные и копирует их в 10-байтовый буфер.

```
#include <stdio.h>
#include <string.h>

void foo(const char* input) {
    char buf[10];
    //Этот трюк позволяет посмотреть стек
    printf("Стек до:   \n%p\n%p\n%p\n%p\n%p\n%p\n\n");
    strcpy(buf, input);
    printf("%s\n", buf);
    printf("Стек после:\n%p\n%p\n%p\n%p\n%p\n%p\n\n");
}
void bar(void) {
    printf("Меня взломали!\n");
}
int main(int argc, char* argv[]) {
    printf("Адрес функции bar = %p\n", bar);
    if (argc != 2) {
        printf("Введите два аргумента!\n");
        return -1;
    }
    foo(argv[1]);
    return 0;
}
```

Рис. 7. Пример программы, уязвимой для атаки на переполнение буфера

Это приложение лучше всего скомпилировать из командной строки, чтобы получить конечную (Release) версию исполняемого файла. Вот что выведет программа, если передать ей строку «Hello» в качестве входного аргумента:

```
C:\Secure>StackOverrun.exe Hello
Адрес функции bar = 00401045
Стек до:
00000000 \
00000000  - локальная переменная buf и выравнивание
7FFDF000 /
0012FF80 <- сохраненный указатель на прежнюю вершину стека
0040108A <- адрес возврата
00410EDE <- указатель на input
Hello
Стек после:
6C6C6548 <- сюда скопировалась сорока "Hello"
0000006F
7FFDF000
0012FF80
0040108A
D0410EDE
```

Если введенная строка будет иметь слишком большой размер, то произойдет переполнение буфера *buf*, а работа программы будет остановлена операционной системой. На экран будет выведена следующая информация:

```
C:\Secure>StackOverrun.exe ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890
Адрес bar = 00401045
Стек до:
00000000
00000000
7FFDF000
0012FF80
0040108A
00410EBE
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567B90
Стек после:
44434241
48474645
4C4B4A49
504F4E4D
54535251
58575655
```

Сообщение операционной системы об ошибке говорит, что произведена попытка выполнить команду по адресу 0x54535251. Это соответствует строке «TSRQ». Подставив вместо этих символов символы, соответствующие адресу *bar* (0x00401045), можно осуществить передачу ей управления в момент перехода по адресу возврата из функции *foo* при ее завершении.

Защита от переполнения буфера

Корректировка исходных кодов программ. Переполнение буфера происходит прежде всего из-за неправильного алгоритма работы программы, который не предусматривает проверок выхода за границы буферов. Также особую роль здесь играют язык программирования C и его стандартные библиотеки. Так как C не содержит средств контроля соответствия типов, то в переменную одного типа можно занести значение другого типа. Стандартные функции C, такие как strcpy, sprintf и gets, работают со строками символов и не имеют в качестве аргументов их размеров. Для решения проблемы рекомендуется замена уязвимых функций на их безопасные аналоги, проверяющие размер строки. Примером библиотеки безопасных строковых функций является библиотека **Microsoft Strsafe**, входящая в состав последних версий компилятора C++ этой фирмы. Небезопасные строковые функции объявлены устаревшими, о чем сообщается при выполнении компиляции в виде предупреждений. Главные свойства безопасных строковых функций:

– размер буфера-приемника обязательно должен передаваться в функцию, чтобы она не выходила за его пределы;

– буферы гарантированно должны содержать завершающий *null,* даже при усечении результата;

– все функции должны возвращать значение типа *HRESULT* с одним кодом успешного завершения – *S_OK*.

Применение проверок выхода за границы. В основе данного метода лежит выполнение проверок выхода за границы переменной при каждом обращении к ней. Это предотвращает все возможные атаки по переполнению буфера, так как полностью исключает само переполнение. Однако у этого решения есть существенный недостаток – значительное (до 30 раз) снижение производительности программы.

Применение проверок целостности. Данный метод изменяет пролог и эпилог всех функций с целью проверки целостности адреса возврата из функции при помощи так называемого «canary word» (кода целостности). Схема защиты изображена на рис. 8.

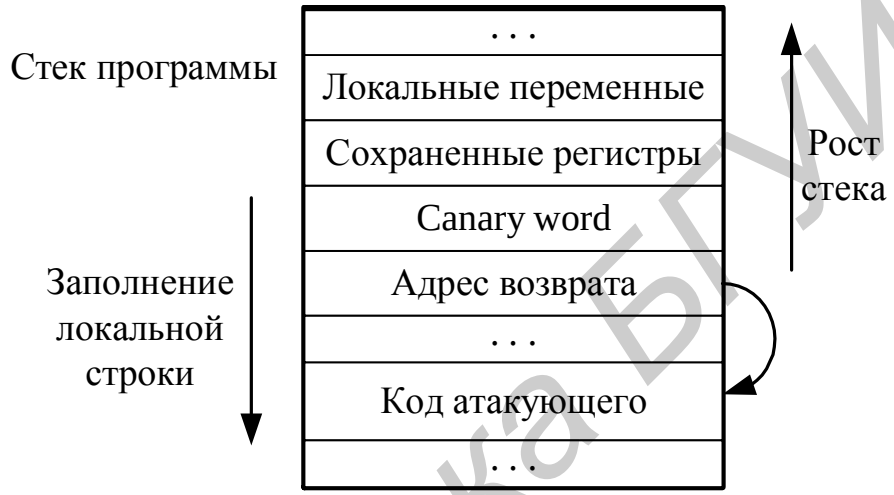

Рис. 8. Внедрение кода проверки целостности в стек программы

Измененный пролог каждой функции выполняет занесение в стек проверочного кода, а эпилог – его проверку, и в случае нарушения программа останавливается с предупреждающим сообщением. При атаке с искажением адреса возврата неизбежно произойдет искажение проверочного кода, что и будет признаком нарушения целостности. Проверочный код заново генерируется при каждом запуске программы и содержит нулевые символы для предотвращения возможности его подбора и обхода злоумышленником. Такая схема защиты реализована в компиляторе Microsoft Visual C++, начиная с версии 7.0 (ключ /GS).

Задания

1. Получите у преподавателя тестовую программу на языке С в соответствии со своим вариантом. Проанализируйте ее на предмет возможной атаки путем несанкционированного вызова функции *hacked* с помощью переполнения строкового буфера. Скомпилируйте программу (в компиляторе Microsoft Visual C++ отключите ключ проверки целостности /GS-) и реализуйте данную атаку. Пример возможной программы для этого задания выглядит следующим образом:

```
#include <stdio.h>
#include <string.h>
```

```
void hacked()
{
  printf("Меня взломали\n");
}
int funct(char *str)
{
  char buf [255];
  int  len = sizeof(buf);
  strcpy(buf, str);
  buf[len - 1] = '\0';
  return(0);
}
int main(int c, char **g)
{
  if (c <= 1) return 0;
  funct(g[1]);
}
```

2. Модифицируйте программу из задания 1 с целью устранения возможности проведения атаки на переполнение буфера с использованием библиотеки безопасных строковых функций Strsafe.

3. Получите у преподавателя тестовую программу повышенной сложности в соответствии со своим вариантом. Проанализируйте ее на предмет возможной атаки типа переполнения буфера. Скомпилируйте программу и реализуйте данную атаку. Пример возможной программы для этого задания выглядит следующим образом:

```
#include <stdio.h>
#include <string.h>

static char message[100];
static char *ptr;
void hacked()
{
  printf("Меня взломали\n");
}
int main(int argc, char **argv)
{
  char a[20];
  ptr = a;
  if (argc < 3)
  {
    printf("Требуется два аргумента\n");
    return -1;
  }
  strcpy(message, argv[1]);
  strncpy(ptr, argv[2], sizeof(a) - 1);
  a[sizeof(a)-1] = '\0';
  return 0;
}
```

6. СТЕГАНОГРАФИЧЕКИЕ МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ

Стеганография занимается разработкой методов передачи секретной информации, в которых скрывается само наличие передачи. В отличие от криптографии, где неприятель точно может определить, является ли передаваемое сообщение зашифрованным текстом, методы стеганографии позволяют встраивать секретные сообщения в безобидные послания так, чтобы невозможно было заподозрить существование встроенного тайного послания. Стеганографическая система, или стегосистема, – совокупность средств и методов, которые используются для формирования скрытого канала передачи информации (рис. 9).

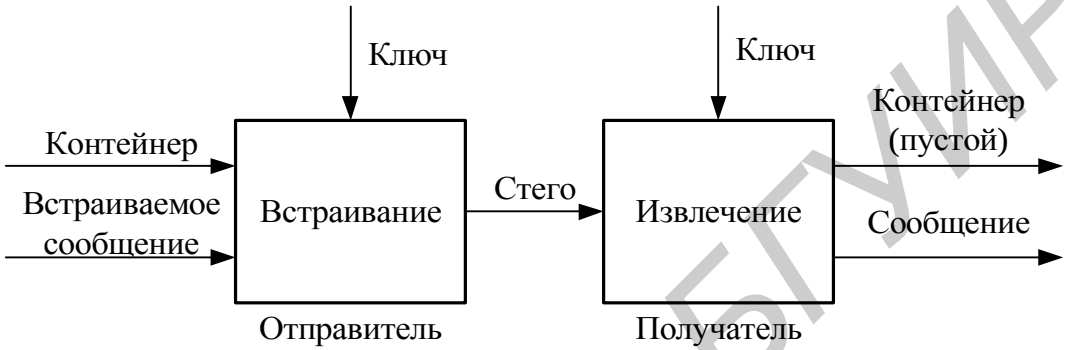

Рис. 9. Обобщенная модель стегосистемы

В настоящее время можно выделить три тесно связанных между собой направления стеганографии: непосредственное сокрытие данных (сообщений), цифровые водяные знаки (watermark) и цифровые отпечатки (fingerprint). Цифровой водяной знак представляет собой некоторую информацию, которая добавляется к цифровому материалу и может быть позднее обнаружена или извлечена для предъявления прав на этот материал. Цифровой отпечаток представляет собой вариацию цифрового водяного знака с тем отличием, что каждая копия защищаемого материала имеет свою собственную уникальную метку, что позволяет впоследствии идентифицировать покупателя, через которого произошло нелицензионное копирование.

Текстовая стеганография

Один из первых методов скрытой передачи текста основан на том, что слова в передаваемом сообщении подбираются так, чтобы вторая буква каждого слова соответствовала очередной букве скрываемого сообщения. Например, в сообщении «Accepted your obertune. Next Friday Aston Shad come away anywhere» спрятано сообщение «cober shown».

Информация в текстовых файлах может кодироваться путем изменения количества пробелов, использования невидимых символов, путем изменения межстрочных интервалов, табуляций и т.д. Но такие стеганографические метки легко удаляются путем простейших атак «print-it-out-type-it-in» («распечатай-набери-заново»).

Менее подвержены взлому системы, основанные на лексической структуре текста, на синонимах языка. И отправитель, и получатель имеют одинаковые

наборы множеств синонимов, имеющих единственное смысловое значение. Например, множество S_0 = {«propensity», «predilection», «penchant», «proclivity»}. Для передачи скрытого текста *M*, используя лексическую стеганографию, выбирается контейнер (текстовый файл) достаточного объема. Сообщение представляется в двоичном виде (*M* => 01000…). Отправитель анализирует слова в контейнере на предмет принадлежности множеству синонимов. Если текущее слово относится к одному из множеств синонимов, то определяется мощность *N* этого множества. Целая часть $\log_2 N$ определяет число битов, которые могут быть закодированы на основании найденного множества синонимов. Например, если текущие 2 бита – 11, а текущее слово – «predilection», то кодирование будет заключаться в замене «predilection» на «proclivity». Для слов с несколькими смыслами подобное кодирование оказывается невозможным. К сожалению, количество синонимов не всегда равно 2^K. Тогда может использоваться система счисления со смешанным основанием.

Метод LSB

Большинство существующих стеганографических систем использует для сокрытия информации метод замены наименее значащих битов в байтах или словах мультимедийных контейнеров (так называемый LSB-метод, от LSB – Least Significant Bit). Данный метод основан на том факте, что при оцифровке изображения или звука всегда присутствует погрешность дискретизации, равная наименьшему значащему разряду числа, определяющему величину цветовой составляющей элемента изображения или амплитуды звукового сигнала. Поэтому замена наименее значащих битов скрытым сообщением в большинстве случаев не вызывает значительной трансформации сигнала и не обнаруживается визуально или аудиально.

Рассмотрим использование данного метода на примере 24-битного растрового RGB-изображения. Одна точка изображения в этом формате кодируется тремя байтами, каждый из которых отвечает за интенсивность одного из трех составляющих цветов: красного (Red), зеленого (Green) и синего (Blue). Интенсивность каждой составляющей лежит в пределах от 0 до 255, то есть каждая составляющая имеет 256 оттенков. Младшие разряды в меньшей степени влияют на итоговое изображение, чем старшие. Из этого можно сделать вывод, что замена одного или двух младших, наименее значащих битов на другие произвольные биты настолько незначительно исказит оттенок пиксела, что зритель просто не заметит изменения (рис. 10). Максимальное количество возможных цветов для этого формата составляет более 16 миллионов. Однако следует иметь в виду, что глаз человека способен различать только около 4 тысяч цветов. Для кодирования этого количества цветов достаточно всего четырех битов $\left(\left\lceil \log_2 \sqrt[3]{4000} \right\rceil = 4\right)$. Как показывает практика, замена одного или двух младших битов не воспринимается человеческим глазом. В случае необходимости можно занять и три разряда, что весьма незначительно скажется на качестве картинки.

а)

R	1	1	0	0	0	0	1	1	195
G	0	0	1	0	0	0	0	0	32
B	0	1	1	0	0	1	1	0	102

б)

R	1	1	0	0	0	0	1	**0**	194
G	0	0	1	0	0	0	**1**	**1**	35
B	0	1	1	0	0	1	**0**	0	100

Рис. 10. Внедрение последовательности из шести битов 101100 в младшие два бита цветовых составляющих одного пиксела RGB-изображения по методу LSB: *а*) – до внедрения; *б*) – после внедрения

Давайте подсчитаем полезный объем такого RGB-контейнера. Занимая два бита из восьми на каждый канал, мы будем иметь возможность спрятать три байта полезной информации на каждые четыре пиксела изображения, что соответствует 25 % объема картинки. Таким образом, имея файл изображения размером 200 Кбайт, мы можем скрыть в нем до 50 Кбайт произвольных данных так, что невооруженному глазу эти изменения не будут заметны.

Метод Patchwork

Данный метод используется для постановки водяных знаков. Он основан на внесении изменений в два участка изображения: на участке A яркость изображения незначительно увеличивается, а на участке B – уменьшается. Рассмотрим основную идею Patchwork на примере изображения, в котором для простоты примем, что все возможные значения яркости пикселей распределены равномерно в диапазоне от 0 до 255.

Выберем на изображении случайным образом две точки A и B, яркость в которых равна a и b соответственно. Теперь положим, что

$$S = a - b.$$

Среднее значение разницы S (обозначим его M_S) после многократного повторения данной процедуры будет равно 0.

Теперь предположим, что описанная процедура повторяется n раз, полагая, что значения a, b и S на i-й итерации равны a_i, b_i и S_i соответственно. Тогда M_S выразится как

$$M_S = \sum_{i=1}^{n} S_i = \sum_{i=1}^{n} (a_i - b_i) = nS \approx 0.$$

Учитывая приведенные выше рассуждения, общий алгоритм встраивания метки может быть представлен следующим образом:

1. Используя оговоренный заранее секретный ключ как начальное значение для криптостойкого генератора псевдослучайных чисел, сгенерировать координаты пары точек (a_i, b_i).
2. Увеличить яркость изображения в точке a_i на значение d , обычно выбираемое в диапазоне от 1 до 5 для изображения с 256 уровнями яркости.
3. Уменьшить яркость изображения в точке b_i на значение d .
4. Повторить шаги 1–3 n раз (n выбирается порядка 10 000).

Модифицированное значение M_S* может быть выражено как

$$M_S^* = \sum_{i=1}^{n}\left((a_i + d) - (b_i - d)\right) = 2dn + \sum_{i=1}^{n}(a_i - b_i) = 2dn + M_S .$$

Таким образом, с каждым новым шагом приведенного выше алгоритма накапливается отклонение на величину $2d$ (рис. 11).

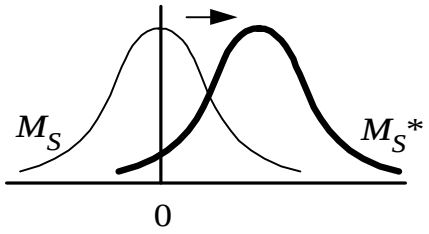

Рис. 11. Сдвиг распределения M_S после внедрения водяного знака

Наличие подобного отклонения от ожидаемого значения свидетельствует о наличии встроенной в изображение метки. Таким образом, владелец может доказать свои интеллектуальные права на изображение, предъявив секретный ключ, который использовался для встраивания метки в изображения.

Задания

1. Реализуйте стеганографическое внедрение сообщений с помощью метода LSB в контейнеры заданного формата (табл. 6, колонка 2). Количество внедряемых в каждую позицию контейнера битов задано в колонке 3.

2. Реализуйте стеганографическую передачу информации с помощью метода генерации текстового контейнера из слов, в которых заданная буква совпадает с внедряемой. Номер буквы задан в колонке 4 табл. 6.

3. Реализуйте стеганографическую систему постановки и проверки цифрового водяного знака с помощью метода Patchwork. Параметры контейнера и метода заданы в колонках 5–7 табл. 6.

Таблица 6

Варианты заданий для работы № 6

№ варианта	Формат контейнера LSB	Количество заменяемых битов	Позиция внедряемой буквы в слове	Формат контейнера Patchwork	d	n
1	2	3	4	5	6	7
1	графический	1	4	аудио	3	10000
2	аудио	2	3	графический	4	20000
3	графический	3	2	аудио	5	30000
4	аудио	1	4	графический	3	10000
5	графический	2	3	аудио	4	20000
6	аудио	3	2	графический	5	30000
7	графический	1	3	аудио	4	20000
8	аудио	2	4	графический	5	30000

7. ЗАЩИТА ОТ НЕЖЕЛАТЕЛЬНОЙ ЭЛЕКТРОННОЙ КОРРЕСПОНДЕНЦИИ

На сегодняшний день доля нежелательной электронной корреспонденции (спама) составляет 60–70 % от всей пересылаемой по сети электронной корреспонденции. Было предложено много антиспамных решений, некоторые из которых были внедрены. Но, к сожалению, эти решения не могут предотвратить рассылку спама, мешающего нормальной работе электронной почты. Проблемы, возникающие из-за спама, выросли от простого раздражения пользователей до существенных проблем безопасности. Наводнение спама приводит к ежегодным убыткам, оцениваемым в сумму до 20 миллиардов долларов, а согласно исследованиям, спам в пределах одной компании приводит к убыткам от 600 до 1 000 долларов ежегодно из расчета на одного пользователя.

В дополнение к потере времени на просмотр и удаление несанкционированных писем, спам также представляет реальную угрозу безопасности. Некоторые вирусы, черви и т.п. типа Melissa, Love Bag и MyDoom используют рассылку спама для своего распространения. Многие спамеры включают в электронные письма злонамеренный код, использующий уязвимость в браузерах, HTML и Javascript. Известно, что некоторые вирусы были разработаны специально для помощи спамерам. Например, червь SoBig занимался поиском открытых прокси-серверов, используемых в дальнейшем для рассылки спама.

Текущие антиспамные решения делятся на четыре основные категории: фильтры, системы обратного поиска, запросы и криптографию. Рассмотрим один из наиболее популярных и эффективных на сегодня методов борьбы со спамом, основанный на фильтрации сообщений с помощью фильтров Байеса.

Фильтры Байеса

Основной принцип работы данного метода заключается в определении принадлежности входящего сообщения к одной из групп (корзин) сообщений на основе анализа составляющих само сообщение слов или слов из служебных полей (заголовков сообщения). Каждая корзина определяется набором входящих в нее слов с вероятностью их появления в данной корзине. Отдельные корзины могут использоваться для нежелательных, личных, деловых и других типов сообщений.

Пусть дано N корзин – B_1, B_2, …, B_N. Требуется определить принадлежность сообщения E к определенной корзине. Воспользуемся формулой Байеса:

$$P(B_i|E) = \frac{P(E|B_i) \times P(B_i)}{P(E)}.$$

Здесь $P(B_i|E)$ – вероятность того, что сообщение E попадет в корзину B_i; это вероятность того, что слова из сообщения E будут принадлежать корзине B_i.

$P(E|B_i)$ – вероятность того, что корзина B_i будет содержать слова из сообщения E.

$P(B_i)$ – вероятность заданной корзины; это вероятность того, что в корзину B_i попадет хоть какое-нибудь сообщение.

$P(E)$ – вероятность существования данного сообщения E.

Для определения того, в какую корзину следует поместить входящее сообщение E, требуется вычислить $P(B_i|E)$ для каждой корзины ($i = 1, 2, \ldots, n$) и выбрать корзину с наибольшим значением этой вероятности. Поскольку во всех этих вычислениях участвует значение $P(E)$, его можно исключить и рассматривать только числитель дроби из выражения формулы Байеса:

$$P(B_i|E) = P(E|B_i) \times P(B_i).$$

Сначала сообщение E разбивается на слова $E_1, E_2, \ldots, E_m$. Затем вычисляется значение вероятности $P(E|B_i)$ как произведение вероятностей $P(E_j|B_i)$ для каждого слова:

$$P(E|B_i) = P(E_1|B_i) \times P(E_2|B_i) \times \ldots \times P(E_m|B_i).$$

Вероятность $P(E_j|B_i)$ равна количеству раз, которое слово E_j ($j = 1, 2, \ldots, m$) попало в корзину B_i при обучении, деленному на общее количество слов в данной корзине. При этом вероятность появления каждого слова рассматривается независимо от других слов (т.е. не учитывается логическая связь слов в предложении), что является необходимым условием для применения формулы Байеса.

Если слово E_j не присутствует в корзине, то вероятность $P(E_j|B_i)$ принимается равной минимальному значению вероятности в этой корзине (для корзин с нежелательными сообщениями) и удвоенному или утроенному значению этой вероятности (для корзин с нормальными сообщениями). Такой ход основан на том предположении, что незнакомые слова по умолчанию не являются признаком спам-сообщения.

Вероятность $P(B_i)$ вычисляется как общее количество слов в корзине B_i, деленное на общее количество слов во всех корзинах.

Окончательная вероятность $P(B_i|E)$ вычисляется как

$$P(B_i|E) = P(E_1|B_i) \times P(E_2|B_i) \times \ldots \times P(E_m|B_i) \times P(B_i),$$

и выбирается корзина с максимальным значением этой вероятности.

Задание

Реализуйте систему фильтрации нежелательной электронной корреспонденции на основе фильтров Байеса. Работа системы должна осуществляться в два этапа – этап обучения и этап фильтрации. На первом этапе формируются начальные значения вероятностей на основе тестовой группы сообщений с заданным признаком принадлежности их к определенной группе фильтрации. На втором этапе система классифицирует входящие сообщения на основе формулы Байеса. В случае ошибки фильтрации следует предусмотреть возможность дообучения системы.

Св. план 2007 г., поз. 61

Учебное издание

Занкович Артем Петрович

ЗАЩИТА ИНФОРМАЦИИ

Практикум
для студентов специальности I – 40 01 01
«Программное обеспечение информационных технологий»
дневной и дистанционной форм обучения

Редактор С. Б. Саченко
Корректор М. В. Тезина

Подписано в печать 2006.	Формат 60x84 1/16.	Бумага офсетная.
Печать ризографическая.	Гарнитура «Таймс».	Усл. печ.л.
Уч.-изд.л. 2,2.	Тираж 100 экз.	Заказ № 625

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330.0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П.Бровки, 6