УДК 004.056

Н.И. Алишов, А.Н. Алишов, А.Я. Бойко, А.В. Громовский, С.В. Зинченко, М.Г. Луцкий, В.А. Марченко, А.В. Палагин, Н.А. Сапунова

ТЕХНОЛОГИЯ СИСТЕМНОЙ ИНТЕГРАЦИИ АППАРАТНЫХ И ПРОГРАММНЫХ СРЕДСТВ ЗАЩИТЫ ИНФОРМАЦИИ

Рассматриваются вопросы создания программно-аппаратного комплекса защиты информации. Приводится описание современных технологий используемых при разработке криптографических средств на базе нераскрываемых шифров. Предложены алгоритмы построения защищенных каналов обмена информации между удаленными системами с реализацией туннельного и транспортного режимов защиты. Описана система «прозрачного» шифрования, которая позволяет минимизировать вмешательство пользователя в свою работу.

Введение

В современных корпоративных информационных системах основной объем циркулирующей информации – это электронные документы [1], которые не отображены на материальных носителях. Это позволяет значительно повысить скорость обработки информации, с одной стороны, и увеличить охват автоматизации повседневной деятельности субъектов организаций до размеров корпорации, целой страны или географического региона, с другой, – делая, таким образом, их применение простым и удобным для любого пользователя. Кроме того, существенным фактором является «неприкосновенность» данных, подлежащих оперативной аналитической обработке, в системах поддержки принятия решений. Однако такие системы требуют полного контроля циркулирующей в них информации, чтобы предотвращать её потери, незаконные изменения или хищения.

Состояние проблемы

Для решения задач контроля и защиты «чувствительной» информации (с точки зрения безопасности) в распределенных системах, согласно последним отчетам ведущих исследовательских организаций, зачастую применяют программно-аппаратные комплексы на базе криптографических алгоритмов [2].

Недостатком таких систем является отсутствие всестороннего учета особенностей среды функционирования и условий применения подсистем комплекса защиты информации. Поэтому при их разработке, как правило, выделяется усредненное ограниченное подмножество наиболее вероятных атак с точки зрения разработчика, которым и должен противостоять разрабатываемый комплекс, без учета конкретных особенностей системы, в которой он функционирует.

Авторами предлагается для построения комплекса защиты использовать уязвимости бизнес-приложений к возможным атакам с учетом особенностей алгоритмов функционирования всей системы. При этом выделяется подмножество атак, которые могут быть реализованы в незащищенной системе, и отсеиваются те, вероятность воспроизведения которых минимальна. Выделенное подмножество атак связано с моделями нарушителей [3], реализующие их. Эти модели нарушителей могут быть представлены в разном виде – математические уравнения, формальные описания и др. [4].

Использование некоторого заданного набора атак и связанных с ними моделей нарушителей позволяет ограничить диапазон применения конкретной реализации программно-аппаратного комплекса защиты информации. Такая проблемная ориентация систем повышает эффективность их реализации, но при появлении новых атак системы становятся «недееспособными», что обусловливает существенные их доработки. На рис. 1 показана предлагаемая формальная модель комп-

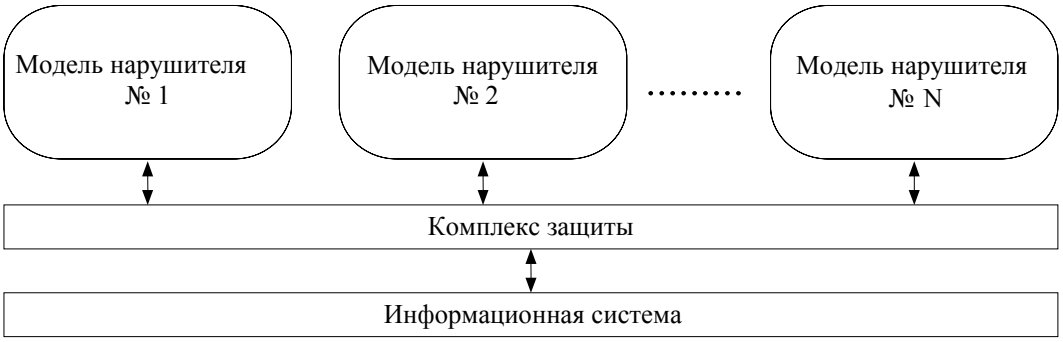


Рис. 1. Формальная модель функционирования комплекса защиты

лекса защиты информации. В данной работе предлагается технология системной интеграции вновь разрабатываемых аппаратных средств с развитыми подсистемами защиты информации учитывая предложенный авторами подход.

Современные подходы к проектированию комплекса защиты

Исходя из указанной формальной модели, процесс проектирования комплекса защиты должен состоять из нескольких этапов:

- разработка и выбор подмножества моделей нарушителей, ориентированных на взаимодействие с комплексом защиты информации;
- выбор существующих архитектурных решений для противодействия заданному подмножеству атак;
- задание области применения комплекса защиты с учетом выбранных моделей нарушителей и требований бизнес-приложений;
- определение конечных характеристик и функциональных возможностей разрабатываемого комплекса.

Этап определения моделей нарушителей является базовым при создании комплекса защиты. Для некоторых систем существуют готовые стандарты, рекомендации и руководящие документы [5, 6], описывающие конкретные наборы угроз или модели нарушителей. В таком случаи, для конечного комплекса защиты информации, необходимыми и достаточными условиями применения является удовлетворение требований, выдвигаемые этими документы.

К сожалению, опыт последних лет показывает, что реализация только этих требований не позволяет гарантировать защиту информации в целом, так как после принятия регламентирующих документов были «изобретены» новые векторы атак на защитные системы, которые не учитываются при разработке таких систем.

Поэтому, зачастую, разработчики при построении новой системы используют актуальные, на данный момент времени, модели нарушителей. Сами модели нарушителей имеют различные представления, но базовым является описание взаимоотношений классов угроз и методов защиты, которые используются в конкретной модели (табл. 1).

Таблица 1. Взаимоотношения угроз и методов защиты

Класс угроз	Метод защиты
Имитация (Spoofing) сообщений (данных)	Аутентификация (Authentication)
Манипулирование (Tampering)	Целостность (Integrity)
Отказ в авторстве (Dispute)	Невозможность отречения (Evidence)
Раскрытие информации (Information Disclosure)	Криптография (Encryption)
Отказ в обслуживании (Denial of Service)	Обеспечение доступности (Availability)
Повышение привилегий (Elevation of Privilege)	Авторизация (Authorisation)

Следует отметить, что в конкретной модели могут использоваться не все классы угроз, и соответственно нет необходимости разрабатывать и реализовывать все методы защиты информации в конкретной прикладной системе [7].

Создание программно-аппаратного комплекса защиты предусматривает использование некоторого набора криптографических средств, которые определяют конечную архитектуру аппаратной составляющей (и в меньшей мере – всего комплекса).

Для реализации криптографических преобразований авторы данной статьи разработали алгоритмы нераскрываемых шифров, использующие небольшое количество алгебраических преобразований при своей работе [8]. На базе этих алгоритмов разработаны опытные образцы микропроцессорного USB-стройства–стеганографа (рис. 2).

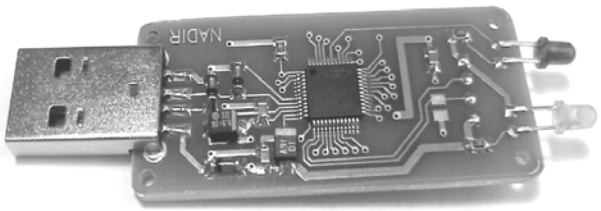


Рис. 2. Опытный образец косвенного стеганографа

Суть предлагаемого метода защиты заключается в следующем. В отправителя и у получателя имеются одинаковые массивы данных, которые являются секретными ключами. Байты информации, подлежащие защите, заменяются (по определенному алгоритму) байтами секретного массива данных. Новый массив данных такого же размера, что и исходное сообщение, передается адресату. Полученный по каналу массив данных подвергается обратному преобразованию: его байты заменяется байтами секретного массива данных (зеркальный алгоритм).

Для создания комплекса защиты информации предлагается следующая функциональная модель (рис. 3).

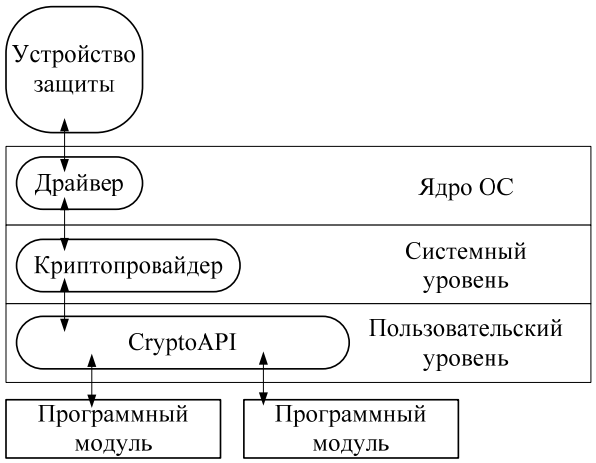


Рис. 3. Модель функционирования комплекса защиты

Как видно из рисунка, при разработке комплекса представляется возможным разделить аппаратную реализацию криптографических преобразований и программную реализацию алгоритмов функционирования и использования, разработанных с учетом выбранного подмножества методов защиты. Базовым элементом взаимодействия между аппаратной и программной частью комплекса защиты информационной системы является криптопровайдер – специализированное прграммное обеспечение сервисного уровня защиты информации [9] (рис. 4).

Криптопровайдер, или Cryptographic Service Provider (CSP), в ОС Windows это библиотека криптографических алгоритмов, доступных прикладным программистам посредством интерфейса CryptoAPI (Cryptographic Application Programming Interface). При взаимодействии с любым криптопровайдером приложения обращаю-тся к функциям CryptoAPI и через опера-ционную систему и ее CryptoSPI (Cryptographic System Program Interface) – к функциям CSP.

Любой криптопровайдер должен экспортировать набор обязательных функций, которые формируют системный программный интерфейс CryptoSPI. При этом каждая из этих функций соответствует некоторым процессам CryptoAPI. Дополнительно любой криптопровайдер должен обеспечивать:

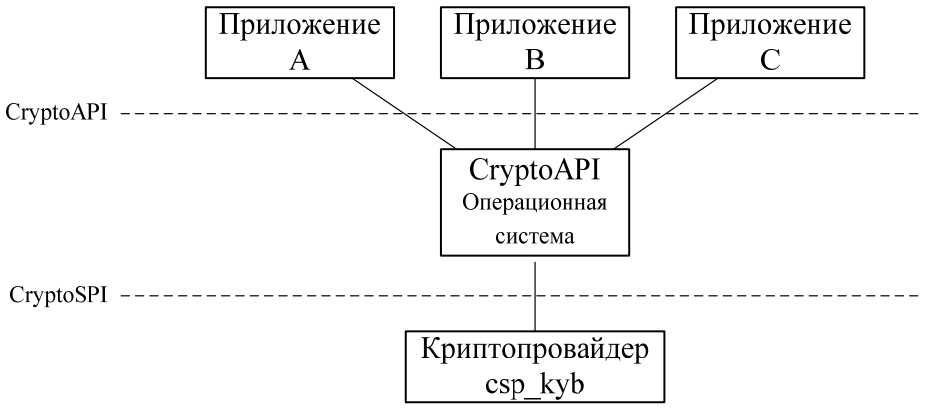


Рис. 4. Схема организации криптопровайдера

- реализацию стандартного интерфейса для взаимодействия с криптопровайдером;
- работу с ключами шифрования, предназначенными для реализации алгоритмов, специфичных для данного криптопровайдера;
- невозможность или усложнение вмешательства третьих лиц в схемы работы алгоритмов.

Реализация вышеуказанных условий позволяет создавать многофункциональные аппаратно-программные комплексы защиты информации с возможностью отделения процессов прикладного программного обеспечения от проблем разработки, адаптации и реализации аппаратно-зависимых частей программных модулей. Таким образом, значительно упрощается создание конечных программных продуктов для систем защиты информации, с одной стороны, а с другой – повышается гибкость разработки и обновления аппаратно-зависимой части комплекса защиты.

При построении криптопровайдера достаточно разработать только специфичный алгоритм, используемый для взаимодействия с устройством, а также необходимые интерфейсные функции. Реализация остальных функций может быть заменена вызовами существующих интерфейсных функций, которые реализованы в других криптопровайдерах как предоставляемых производителем ОС, так и разработанные сторонними разработчиками.

Для идентификации выбранного алгоритма в системе используется строка символов в специальном формате. Таким образом, используя один криптопровайдер, можно реализовать несколько различных версий функционирования алгоритма (аппаратная реализация, программная реализация и т.п.). Листинг подобно реализации представлен ниже. Весомую часть потерь «чувствительных» данных составляет непреднамеренная утечка информации, допущенная по незнанию или невнимательности пользователя.

Для предотвращения таких наруше-

```
Decrypt(HCRYPTKEY hKey, HCRYPTHASH hHash, BOOL Final, DWORD dwFlags,
BYTE*pbData, DWORD*pdwDataLen)
{       // Распаковка структуры ключа
        KEY_INFO *pKey = (KEY_INFO*) hKey;
        if ( pKey->algId == CALG_KIB_CRYPT)
        {       // Реализация модификации алгоритма        return TRUE;   }
        if ( pKey->algId == CALG_KIB_CRYPT_USB)
        {       // Реализация модификации алгоритма        return TRUE;   }
          // Вызов реализации из другого криптопровайдера
          return CryptDecrypt(hKey, hHash, Final, dwFlags, pbData, pdwDataLen);
}
```

ний необходимо максимально упростить алгоритмы использования системы защиты либо вообще лишить пользователя возможности управлять средством защиты, чтобы исключить принятие необдуманного решения и выполнение нерегламентированных операций.

Для решения этой задачи предлагается создать «прозрачную» систему защиты, которая будет требовать минимального вмешательства пользователя при функционировании.

Перед рассмотрением принципов работы «прозрачной» системы шифрования необходимо кратко и упрощенно рассмотреть принцип вызова системных функций, на примере MS Windows, для работы с файлами и методы их перехвата на уровне ядра пользователя. Вопросы перехвата файловых операций на уровне системного ядра в данной статье не рассматриваются.

Существует два способа вызова системных функций прикладными программами (рис. 5):

1) раннее связывание (статически импортируемые функции). Этот метод основан на том, что компилятору известен перечень импортируемых программой функций. Опираясь на эту информацию, компилятор формирует так называемую таблицу импорта ряда функций исполняемого файла. Таблица импорта – это особая структура, которая содержит список используемых программой библиотек и список импортируемых из каждой библиотеки функций. Для каждой функции в таблице имеется поле для хранения адреса, но на стадии компиляции адрес неизвестен. В процессе загрузки исполняемого файла система анализирует его таблицу импорта, загружает все перечисленные в ней DLL и заносит в таблицу импорта реальные адреса функций этих DLL;

2) позднее связывание. Отличается от раннего связывания тем, что загрузка DLL производится динамически при помощи функции API LoadLibrary, которая позволяет программе загрузить интересующую ее библиотеку в любой момент времени. Соответственно для получения адреса функции применяется функция kernel32.dll GetProcAddress.

Соответственно перехват требуемой функции можно выполнить тремя основными методами:

– модификация таблицы импорта требуемого приложения;

– перехват функций LoadLibrary и GetProcAddress;

– модификация кода перехватываемой функции.

Первые два метода имеют существенные недостатки [10], поэтому на практике применяется третий (рис. 6), самый сложный в реализации, но не имеющий недостатков, свойственных первым двум методам.

Модификация кода перехватываемой функции позволяет выполнять перехват вызовов системных функций приложения совершенно прозрачно, но для обеспечения безопасности данных требует применения дополнительных подходов, обеспечивающих ограничение для приложений на доступ к зашифрованным данным. Одним из перспективных способов реализации такого ограничения может служить вариант многоуровневой электронной подписи приложения, что позволяет разграничить доступ приложений к

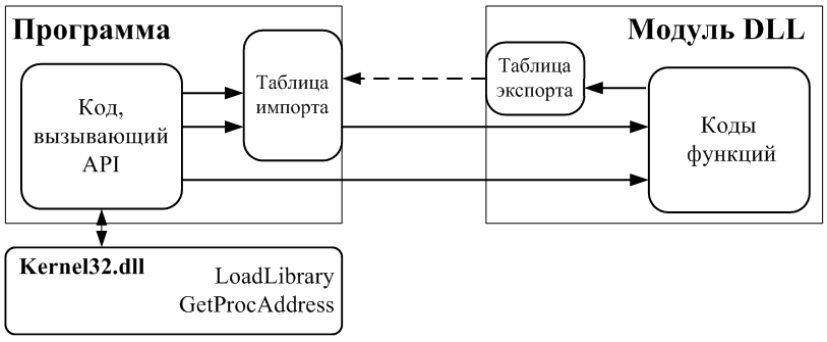


Рис. 5. Способ вызова системных функций прикладными программами

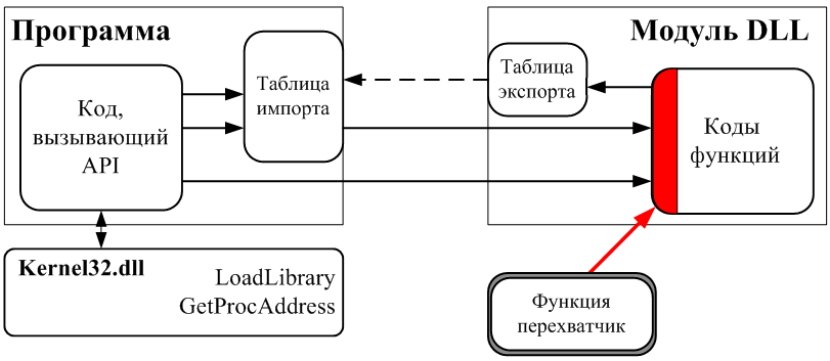


Рис. 6. Модификация кода перехватываемой функции

защищенным данным пользователя, предоставляя доступ только тем из них, которые подписаны пользователем. В частности, такой способ делает невозможной «утечку» защищаемых данных с помощью программных закладок, которые могут активизироваться при работе пользователя в системе.

В разработанном комплексе используется трехуровневая система подписывания приложения:

1) подпись исполняемого файла приложения (позволяет избежать подмены исполняемого файла злоумышленником);

2) подпись образа в памяти исполняемого приложения (позволяет избежать модификации кода приложения злоумышленником);

3) подпись функций работы с защищенными данными (позволяет избежать модификации отдельных функций злоумышленником).

Для обеспечения прозрачности работы с защищаемыми данными необходимо осуществить перехват некоторых системных функций работы с файлами, в частности:

- CreateFile
- CloseHandle
- DeleteFile
- ReadFile
- WriteFile
- CreateFileMapping
- UnmapViewOfFile
- FileSeek

Набора этих функций достаточно для обеспечения нормального функционирования приложения при работе с защищаемыми данными. Отдельно стоит упомянуть, что при некорректном использовании этих функций могут возникать проблемы. В частности, любой объект файлового отображения, созданный с помощью функции MapViewOfFile, должен закрываться с помощью функции UnmapViewOfFile. Если это условие не выполнено, то часть данных может быть потеряна. При этом используя только системные функции, без работающей системы перехвата, даже при несоблюдении вышеуказанного условия данные в файл будут записаны без вызова функции UnmapViewOfFile, после вызова функции закрытия файла. Но этот режим работы системы, является не стандартным, а потому не поддерживается.

Приведем пример реализации перехваченных функций чтения файла (ReadFile) и создания файлового отображения (MapViewOfFile).

```
function AN_ReadFile_(
        hFile: THandle; var Buffer;
nNumberOfBytesToRead: DWORD;
        var lpNumberOfBytesRead:
DWORD; lpOverlapped: POverlapped
        ): BOOL; stdcall;
    var file_:PfRecord;
      fpos:DWORD;
    begin
    AN_ReadFile_ := TReadFile(

@functions[xTReadFile].AC_fnk)
        ( hFile, Buffer, nNum-
berOfBytesToRead,
        lpNumberOfBytesRead,
lpOverlapped );
     EnterCriticalSection(g_lock);
     if (Result) then
     begin
      fileobj_rec_Check();
file_ := fileobj_rec_Find(hFile,-1,-1,-1);
```

```
        if file_<>nil then
        begin
fpos := DWORD(FileSeek(hFile,0,1)) -
lpNumberOfBytesRead;
DecryptBuffer_inline(@buffer, fpos,
nNumberOfBytesToRead);
        end; //else not crypted
       end; //error read data
      LeaveCriticalSection(g_lock);
      end;

      function AN_MapViewOfFile_(
        hFileMappingObject: THandle;
dwDesiredAccess: DWORD;
dwFileOffsetHigh,dwFileOffsetLow,
dwNumberOfBytesToMap: DWORD
): Pointer; stdcall;
      var map_orig:Pointer;
       map_, mapd_:PfRecord;
      begin
map_orig := TMapViewOfFile(
@functions[xTMapViewOfFile].AC_fnk)
          (hFileMappingObject, dwDe-
siredAccess, dwFileOffsetHigh, dwFi-
leOffsetLow, dwNumberOfBytesToMap);
       EnterCriticalSection(g_lock);
      if map_orig<>nil then
       begin
        fileobj_rec_Check();
        map_:=fileobj_rec_Find(-
1,hFileMappingObject,-1,-1);
        if map_<>nil then
        begin
mapd_:=fileobj_rec_Add(-1,-1,0,-1);
mapd_._type := 2;
mapd_._map_data.data_size := dwNum-
berOfBytesToMap;
mapd_._map_data.data_orig := map_orig;
mapd_._map_data.data_new := VirtualAl-
loc(nil,dwNumberOfBytesToMap,
MEM_COMMIT,PAGE_READWRITE);
map_orig := mapd_._map_data.data_new;
mapd_.prm3 := Integer(map_orig);
        CopyMemory(
        mapd_._map_data.data_new,
        mapd_._map_data.data_orig,
        mapd_._map_data.data_size);
        DecryptBuffer_inline(
        mapd_._map_data.data_new, dwFileOff-
setLow, mapd_._map_data.data_size);
end; //else not crypted file
end else
        begin
        end;
       LeaveCriticalSection(g_lock);
       AN_MapViewOfFile_:= map_orig;
      end;
```

Схема работы алгоритма, перехвата функций представленная на рис. 7.

Таким образом, пользователь, работая в системе с документами, не имеет возможности использовать комплекс защиты с неразрешёнными приложениями, что значительно усложняет взлом, а также минимизирует вероятность потери информации или её несанкционированное распространение. Пример интерфейса программы показан на рис. 8.

Современные корпоративные приложения представляют собой распределённые системы, взаимодействующие с помощью сетевых коммуникаций. Соответственно и защищаемые документы или информационные сообщения, циркулирующие внутри таких систем, передаются по сети. Поэтому в аппаратно-программных комплексах защиты необходимо реализовать алгоритмы, учитывающие сетевые особенности передачи защищаемой информации и максимально противодействующие перехватам данных нарушителями, находящимися в той же или транзитной сети.

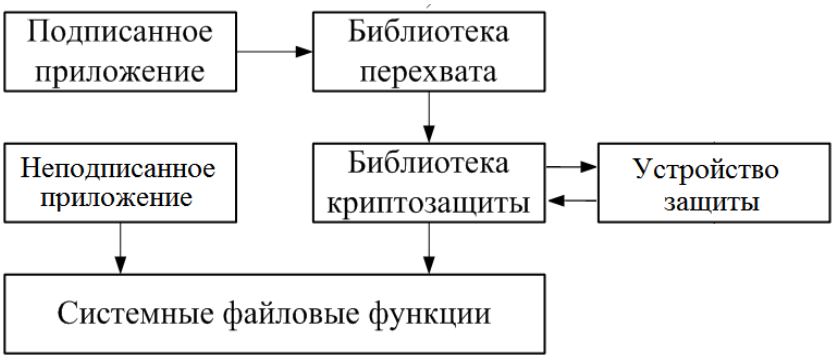


Рис. 7. Схема работы алгоритма перехвата

Для решения указанных задач представляется целесообразным создавать систему защиты сети в целом, а не ограничиваться защитой приложений. Другими словами, система защиты сети должна действовать на сетевом уровне модели OSI. Преимущество такого подхода очевидно, так как передача данных не может быть осуществлена в обход IP-протокола, независимо от физической среды, технологии канального уровня и протоколов более высокого уровня. Это гарантирует защиту всех сетевых приложений, без их модификации.

Решение задач защиты информации в сети предусматривает реализацию авторизации и аутентификации пользователей в удаленных системах, а также защиты виртуального канала передачи информации от прослушивания, дублирования и подмены сообщений.

Современные стандарты [11] и рекомендации [12] для целей удаленной аутентификации и авторизации используют специализированный протокол Kerberos – сетевой протокол, позволяющий безопасно передавать данные для идентификации через незащищённые сети. В существующих реализациях систем защиты, использующих аппаратные части для криптографических преобразований, сами ключи для протокола аутентификации хранятся в зашифрованном виде во флеш-памяти устройства. После записи этих ключей в устройство они никогда не покидают его пределов и поэтому не могут быть скопированы на другой носитель.

Для защиты виртуального канала передачи информации авторами комплекс разработан специализированный программный KeyTunnel, реализующий шифрование сетевого трафика между узлами с использованием аппаратного устройства шифрования (АУШ-стеганограф) – электронного USB-ключа и созданного для него криптопровайдера. Безопасность передаваемых данных в KeyTunnel обеспечивает модифицированный протокол IPSec (modIPSec), в который, помимо стандартных решений по безопасности, включены дополнительные функции для работы с аппаратным ключом и криптопровайдером.

IPSec обеспечивает шифрование всего IP-трафика между двумя узлами, или двумя сетями, или сочетаниями хостов с возможным дифференцированием оконечных точек для различных служб безопасности. Ключи могут быть созданы вручную или при помощи протокола Internet Key Exchange (IKE). Ключи являются частью однонаправленной подсистемы Security Association, которая определяет IP-адрес назначения, ключи и алгоритмы шифрования, а также используемый «Протокол». «Протокол» здесь соответствует одному из двух режимов обеспечения безопасности IPSec: Authentication Header (AH) и Encapsulated Security Payload (ESP). В AH код аутентификации сообщения (message authentication code, MAC) создается для заголовков IP-пакетов. Это позволяет приложению-получателю обнаружить поддельные или измененные IP-пакеты. Однако полезные данные представлены в открытом тексте и доступны любому, в том числе и перехватившему IP-пакеты злоумышленнику.

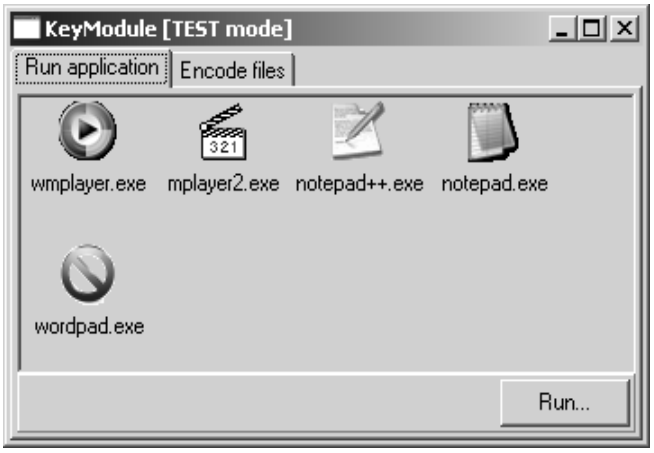


Рис. 8. Интерфейс программы со списком пользовательских приложений

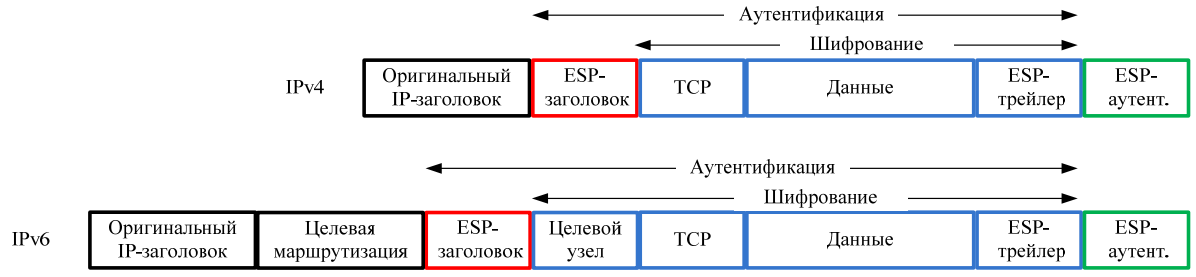
ESP обеспечивает конфиденциальность и целостность полезных данных IP-пакета, но не заголовка. Использование этих двух протоколов в вышеописанном порядке называется транспортным режимом. Кроме того, возможно их использование в туннельном режиме, когда оригинальный IP-пакет туннелируется в другой IP-пакет. В этом случае к полезным данным, готовым к передаче по виртуальной частной сети, IPSec добавляет заголовок для идентификации защищенных пакетов и, перед передачей по сети, инкапсулирует их в другие IP-пакеты (рис. 9).

Для защиты высокоуровневых соединений в Веб-браузерах с целью безопасной передачи данных, особенно в приложениях электронной коммерции для банковских и других конфиденциальных сделок, используется протокол SSL (Secure Socket Layer; ему на замену принят новый стандарт TLS(Transport Layer Security)). Пользователь аутентифицируется через Веб-браузер при помощи паролей или других методов, используя сервер. Он, в свою очередь, аутентифицируется браузером через соответствующие цифровые сертификаты. Это дает гарантию того, что сделка происходит между легитимным сайтом и легитимным пользователем.

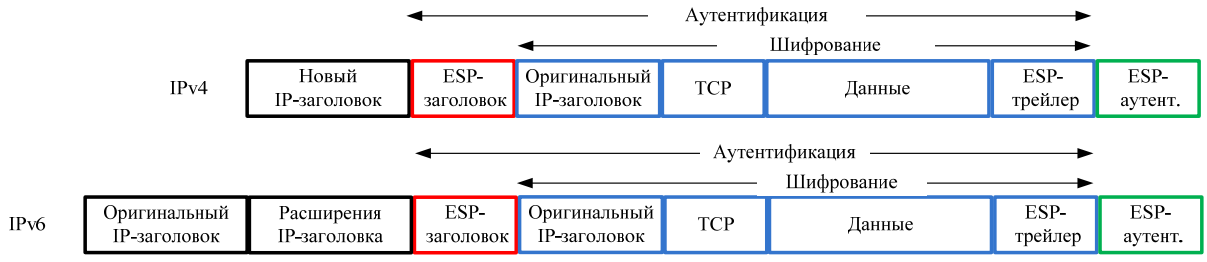
Следует обратить внимание, что сам факт использования SSL не является гарантией подлинности сервера, так как протокол SSL может применять абсолютно любой сайт или сервер. Аутентифицирует сервер информация, содержащаяся в цифровом сертификате, который подтверждает подлинность сервера. Цифровой сертификат содержит публичный ключ сервера, подписанный службой сертификации. Браузер создает случайный ключ, шифрует его с помощью открытого ключа сервера и отправляет на сервер. Этот ключ применяется для генерации ключей (как на сервере, так и в браузере) которые используются при блочном или потоковом шифровании.

По сути, IPSec работает на сетевом, т. е. на третьем уровне. В результате передаваемые IP-пакеты будут защищены прозрачным для сетевых приложений и инфраструктуры образом. IPSec, в отличие от SSL, который работает на транспортном (т. е. четвертом) уровне и теснее связан с более высокими уровнями модели OSI, обеспечивает низкоуровневую защиту.

Разработанные программные комплексы KeyTunnel и modIPSec реализуют



а – транспортный режим



б – туннельный режим

Рис. 9. Режимы IPSec. ESP шифрование и аутентификация

защиту передаваемой информации в двух режимах: транспортном и туннельном. В транспортном режиме KeyTunnel шифрует полезные данные IP- протокола, при этом сами IP-заголовки остаются в открытом виде для их использования при маршрутизации трафика, как показано на рис. 10.

В туннельном режиме KeyTunnel использует инкапсуляцию пакетов между шлюзами (IP_B и IP_C) двух виртуальных частных сетей (VPN) (рис. 11).

Алгоритм туннелирования состоит из следующих действий:

1) Между шлюзами IP_B и IP_C создается туннель.

2) IP-пакеты шифруются с помощью АУШ.

3) Эти пакеты затем инкапсулируются в другой пакет IP и переадресуются на вход в туннель IP_B или выход из туннеля IP_C.

4) На выходе из туннеля (VPN-шлюз обслуживает IP_D) оригинальный пакет будет извлечен, расшифрован (с помощью АУШ) и отправлен на целевой узел (IP_D).

Используемый подход позволяет KeyTunnel не только создавать защищенные сетевые соединения между пользователями, но и объединять изолированные сети через общую инфраструктуру для виртуальных сетей.

Функциональность протокола modIPSec в KeyTunnel основана на механизме прокси-безопасности [14, 15] (рис. 12), отвечающем за передачу IP-пакетов в другой прокси-сервер, который в свою очередь подтверждает и расшифровывает их перед передачей на целевой узел.

Прокси-сервер безопасности упрощает реализацию взаимодействия между узлами сети. Иными словами, прокси-безопасность позволяет выполнять шифрование данных в несколько потоков от нескольких запросов. В этом режиме IP-пакеты из потока данных группируются, шифруются и направляются на удаленный прокси-сервер через туннель.

Для пользователей процедуры защиты окажутся столь же прозрачными, как и сам протокол IP.

Достаточно обеспечить поддержку модифицированного протокола IPSec на обоих концах соединения с помощью программного комплекса KeyTunnel; промежуточные сетевые узлы могут вообще ничего «не знать» о modIPSec.

Пример инициализации прокси-сервера на языке C# приведен на следующем листинге:

```
        HttpListener listener = new
HttpListener();
        int port = 8086;
        // Создание префикса HttpLis-
tener
        string prefix =
string.Format("http://*:{0}/", port);
        listener.Prefixes.Add(prefix);
        // Запуск прослушивания
HttpListener
        listener.Start();
while (true)
{
        HttpListenerContext request =
        _listener.GetContext();
        Обработка запроса
        ProcessRequest(request);
}
```

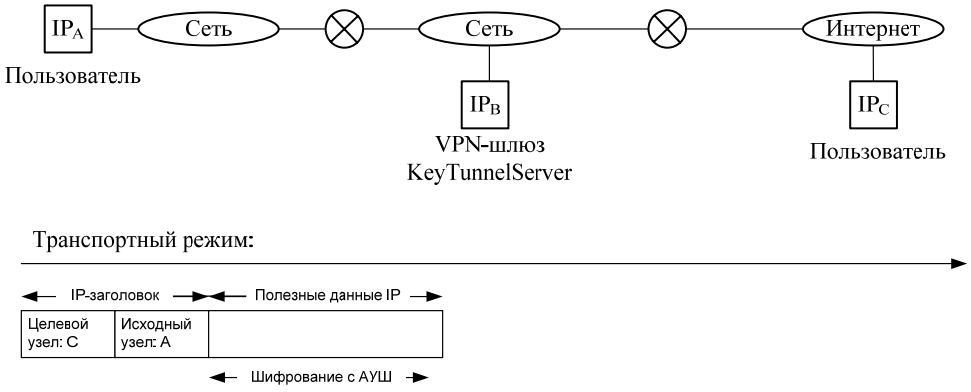


Рис. 10. Транспортный режим модифицированного протокола IPSec

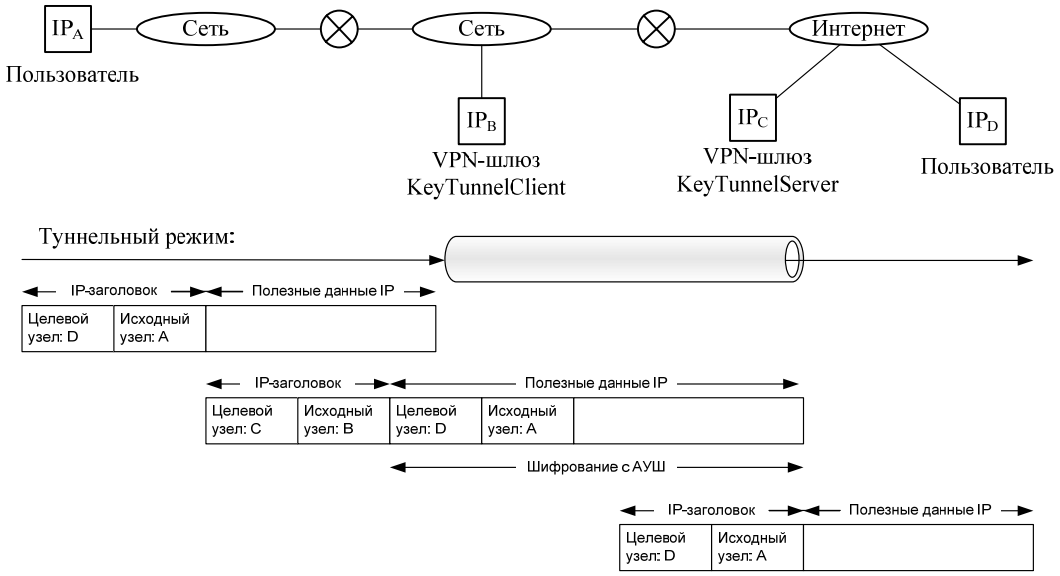


Рис. 11. Туннельный режим модифицированного протокола IPSec

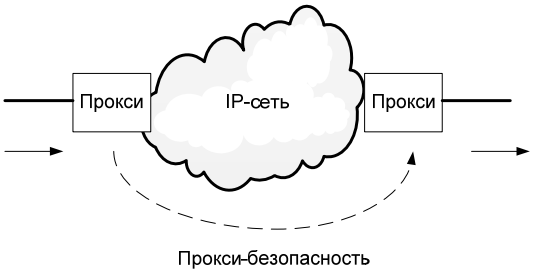


Рис. 12. Механизм прокси-безопасности

Механизм обработки запросов прокси-сервера представлен на следующем листинге:

```
// Передача данных из запроса HttpListenerContext в HttpWebRequest
buffer = new byte[256];
Stream instream = context.Request.InputStream;
int incount = instream.Read(buffer, 0, buffer.Length);
while (incount > 0) {
psRequest.GetRequestStream().Write(buffer, 0, incount);
incount = instream.Read(buffer, 0, buffer.Length); }
instream.Close();
// Отправка запроса на удаленный сервер и прием ответа
psResponse = (HttpWebResponse)psRequest.GetResponse();
//Конвертирование HttpWebResponse в HttpListenerContext и отправка данных пользователю
foreach (string key in psResponse.Headers.Keys){
switch (key)
{
case "Transfer-Encoding":
context.Response.SendChunked = (psResponse.Headers[key].ToLower() == "chunked") ? true : false; break;
case "Content-Length":
context.Response.ContentLength64 = psResponse.ContentLength;
break;

case "Content-Type":
context.Response.ContentType = psResponse.Headers[key];
break;

case "Keep-Alive":
context.Response.KeepAlive = true;
break;

default:
context.Response.Headers.Add(key, psResponse.Headers[key]);
break; } }
// Передача данных из запроса HttpWebResponse
buffer = new byte[256];
Stream outstream = psResponse.GetResponseStream();
int outcount = outstream.Read(buffer, 0, buffer.Length);
```

```
            while (outcount > 0)    {
            con-
text.Response.OutputStream.Write(buffer
, 0, outcount);
            outcount = out-
stream.Read(buffer, 0, buffer.Length);
}
            // Закрытие потока
            outstream.Close();
context.Response.OutputStream.Close();
```

Структура программного комплекса KeyTunnel имеет следующий вид.

1. ПО для организации VPN-шлюза.
2. Интерфейс администратора (рис. 13): KeyTunnelServer.
3. Библиотека реализации modIPSec и прокси-безопасности.
4. Служба организации туннеля: kss.
5. Библиотека взаимодействия с АУШ.
6. Драйверы для АУШ.

Пользовательское ПО.

1. Интерфейс пользователя (рис. 14): KeyTunnelClient.
2. Библиотека реализации IPSec и прокси-безопасности: KeyTunnel.dll.
3. Служба прокси-перехвата HTTP (перехват запросов пользователя браузера): ksp.
4. Библиотека взаимодействия с АУШ.
5. Драйверы для АУШ.

Схема перехвата http-запросов пользователя в созданном с помощью комплекса KeyTunnel защищенном канале передачи данных с Web -узлом представлена на рис. 15.

Таким образом, использование разработанного комплекса защиты не требует внесения каких-либо изменений в существующее или разрабатываемое ПО, так как использует стандартные механизмы взаимодействия.

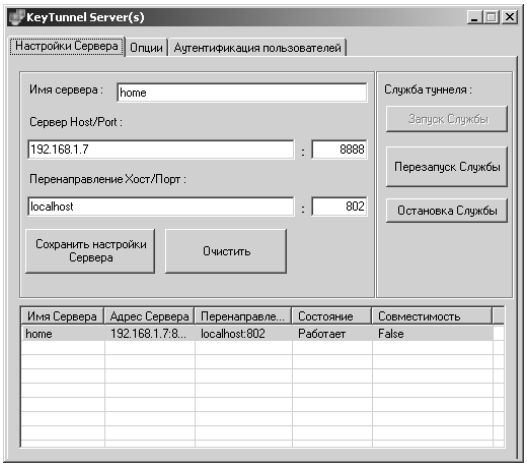


Рис. 13. Интерфейс администратора KeyTunnelServe

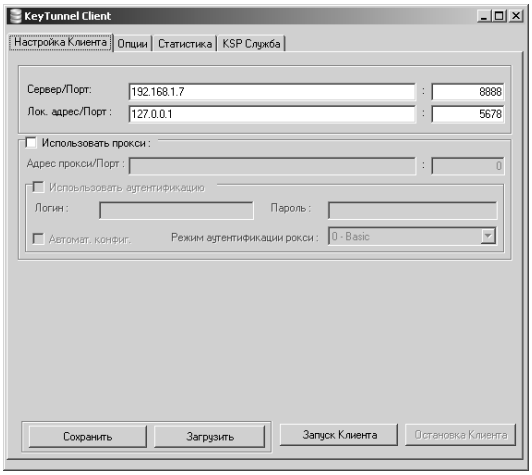


Рис. 14. Интерфейс пользователя KeyTunnelClient

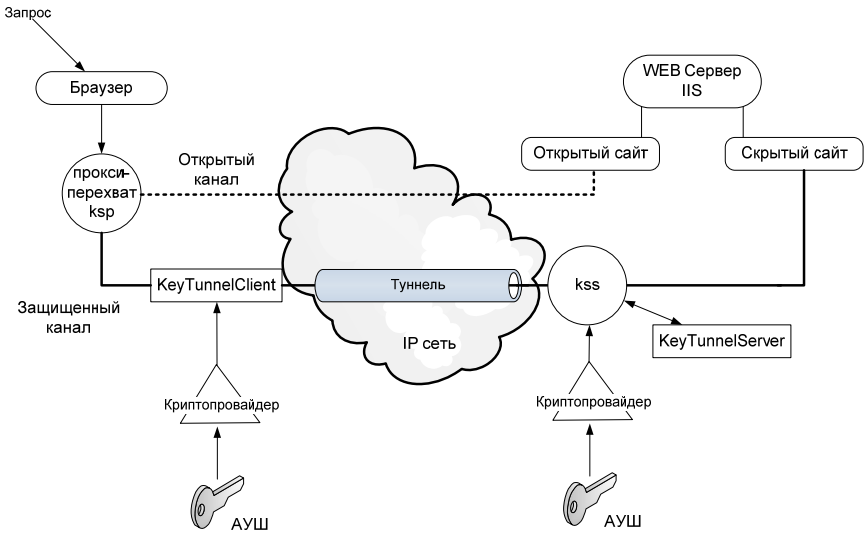


Рис. 15. Организация защищенного канала передачи данных со скрытым Web-узлом

Заключение

Разработка программно-аппаратных комплексов защиты с учетом моделей нарушителей является перспективным направлением исследований в области информационной безопасности. Реализация описанных режимов работы программно-аппаратных комплекса позволяет строить новые подсистемы защиты с учетом основных возможных атак для защищаемой информационной системы, что значительно упрощает этап разработки таких подсистем с точки зрения архитектурных особенностей. Разработанный аппаратно-программный комплекс прошел экспериментальные исследования в настольных компьютерах, в локально-корпоративной сети компьютеров, а также в глобальной сети Интернет.

1. *ДСТУ* 2732:2004. Діловодство й архівна справа. Терміни та визначення понять. – К., 2004 – 36 с. (Державний стандарт України).
2. *Getgen K*. Encryption and Key Management Industry Benchmark Report. – http:// www.trustcatalyst.com 2009. – 33 p.
3. *Dolev D., Yao A.C*. On the security of public key protocols // Proc. of the IEEE 22nd Annual Symposium on Foundations of Computer Science. – 1981. – P. 350–357.
4. *Swiderski F., Snyder W*. Threat Modeling. – Microsoft Press, 2004. – 288 p.
5. *Virtue T. M*. Payment Card Industry Data Security Standard Handbook. – John Wiley, 2008. – 216 p.
6. *BS* 10012:2009 Data protection. Specification for a personal information management system. – British Standards Institution, 2009. – 32 p.
7. *Uncover* Security Design Flaws Using The STRIDE Approach / Shawn Hernan, Scott Lambert, Tomasz Ostwald, Adam Shostack // MSDN Magazine. – November 2006. – P. 68–75.
8. *Алишов Н.И., Марченко В.А., Оруджева С.Г.* Косвенная стеганография как новый способ передачи секретной информации // Комп'ютерні засоби, мережі та системи. – 2009. – № 8. – С. 105–112.
9. *Cryptographic* Service Providers. – http://msdn.microsoft.com/en-us/library/aa380245%28VS.85%29.aspx
10. *Перехват* API функций в Windows. – http://www.wasm.ru/article.php?article=apihook_1
11. *ISO/IEC* 9594. The Directory: Overview of concepts, models and services. – Geneva, ISO, 2008. – 23 p. – (International standart).
12. *J. Kohl, B. Neuman, T. Ts'o*. The Evolution of the Kerberos Authentication Service // Distributed Open Systems. – IEEE Computer Society Press. – 1994. – P. 78–94.

13. *The Kerberos* Network Authentication System (RFC4120) / C. Neuman, T. Yu, S. Hartman, K. Raeburn. – 2005. – http://www.ietf.org/rfc/rfc4120.txt
14. *Goto A*. Safe and Secure Ubiquitous Communication. – international workshop on network security and wireless communications 27 Jan 2005.– http://www.it.ecei.tohoku.ac.jp/~kato/workshop2005/NTT-goto-slides.pdf
15. *Network* Security: Know It All / J. Joshi, S. Bagchi, B.S. Davie et al. – Morgan Kaufmann, 2008. – 368 p.

Получено 23.04.2010

Об авторах:

Алишов Надир Исмаил-оглы,
доктор технических наук,
старший научный сотрудник,

Алишов Анар Надир-оглы,
инженер-программист,

Бойко Александр Яковлевич,
научный сотрудник,

Громовский Александр Владимирович,
младший научный сотрудник,

Зинченко Сергей Викторович,
младший научный сотрудник,

Луцкий Максим. Григорьевич,
кандидат технических наук,
проректор,

Марченко Виталий Анатолиевич,
кандидат технических наук,
научный сотрудник,

Палагин Александр Владимирович,
доктор технических наук,
зам. директора института,

Сапунова Надежда Александрова,
ведущий инженер-программист.

Место работы авторов:

Институт кибернетики
имени В.М. Глушкова НАН Украины,
03187, Киев-187,
Проспект Академика Глушкова, 40.
Проспект Комарова, 1.
Тел. 526 3427