

ГОУВПО
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра программной инженерии

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ
к лабораторным работам по дисциплине
«Безопасность программ и данных»

(для студентов направления подготовки 09.03.04 «Программная инженерия »)

Донецк-ДонНТУ-2020

ГОУВПО
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра программной инженерии

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ
к лабораторным работам по дисциплине
«Безопасность программ и данных»

(для студентов направления подготовки 09.03.04 “Программная инженерия”)

Рассмотрено на заседании кафедры
программной инженерии

Протокол № 1 от 28.08.2020

Утверждено на заседании

учебно-издательского Совета ДонНТУ
протокол № от

Донецк –2020

Методические указания и задания к лабораторным работам по дисциплине «Безопасность программ и данных» для студентов направления подготовки 09.03.04 «Программная инженерия», Сост.: Чернышова А.В., Донецк, ДонНТУ, 2020 - 113 стр.

Приведены методические указания и задания к выполнению лабораторных работ по дисциплине «Безопасность программ и данных» для студентов направления подготовки 09.03.04 «Программная инженерия». Излагаются вопросы, связанные с криптографическими методами защиты информации, защитой авторских программ от несанкционированного использования, защитой ресурсов на уровне операционных систем. Рассматриваются простейшие базовые криптографические алгоритмы, базовые симметричные и асимметричные криптоалгоритмы, реализация криптографических средств защиты с использованием криптографических библиотек, использование простейших стеганографических средств защиты информации, средства обфускации программ.

Методические указания предназначены для усвоения теоретических основ и формирования практических навыков по курсу «Безопасность программ и данных».

Составители: ст.преп. каф. ПМиИ Чернышова А.В.

Рецензент: доцент кафедры КМД Губенко Н.Е.

Лабораторная работа №1

Тема: Изучение возможностей программных закладок, логических «бомб» и «троянских коней».

Цель: Изучение принципов работы программных закладок, логических «бомб» и «троянских коней». Провести анализ этих программ и представить модели реализации с целью изучения возможностей взлома компьютерной системы. Анализ средств защиты от перечисленных программ. Изучение возможности использования логических «бомб» с целью защиты от незаконного использования авторских программ.

Методические указания к лабораторной работе

Современная концепция создания компьютерных систем предполагает использование программных средств различного назначения в едином комплексе. К примеру, типовая система автоматизированного документооборота состоит из операционной среды, программных средств управления базами данных, телекоммуникационных программ, текстовых редакторов, антивирусных мониторов, средств для криптографической защиты данных, а также средств аутентификации и идентификации пользователей. Главным условием правильного функционирования такой компьютерной системы является обеспечение защиты от вмешательства в процесс обработки информации тех программ, присутствие которых в компьютерной системе не обязательно. Среди подобных программ, в первую очередь, следует упомянуть компьютерные вирусы. Однако имеются вредоносные программы еще одного класса. От них, как и от вирусов, следует с особой тщательностью очищать свои компьютерные системы. ***Это так называемые программные закладки, которые могут выполнять хотя бы одно из перечисленных ниже действий:***

- вносить произвольные искажения в коды программ, находящихся в оперативной памяти компьютера (программная закладка первого типа);

- переносить фрагменты информации из одних областей оперативной или внешней памяти компьютера в другие (программная закладка второго типа);

- исказить выводимую на внешние компьютерные устройства или в канал связи информацию, полученную в результате работы других программ (программная закладка третьего типа).

Программные закладки можно классифицировать и по методу их внедрения в компьютерную систему:

- программно-аппаратные закладки, ассоциированные с аппаратными средствами компьютера (их средой обитания, как правило, является BIOS — набор программ, записанных в виде машинного кода в постоянном запоминающем устройстве — ПЗУ);

- загрузочные закладки, ассоциированные с программами начальной загрузки, которые располагаются в загрузочных секторах (из этих секторов в процессе выполнения начальной загрузки компьютер считывает программу, берущую на себя управление для последующей загрузки самой операционной системы);

- драйверные закладки, ассоциированные с драйверами (файлами, в которых содержится информация, необходимая операционной системе для управления подключенными к компьютеру периферийными устройствами);

- прикладные закладки, ассоциированные с прикладным программным обеспечением общего назначения (текстовые редакторы, утилиты, антивирусные мониторы и программные оболочки);

- исполняемые закладки, ассоциированные с исполняемыми программными модулями, содержащими код этой закладки (чаще всего эти модули представляют собой пакетные файлы, т. е. файлы, которые состоят из команд операционной системы, выполняемых одна за одной, как если бы их набирали на клавиатуре компьютера);

- закладки-имитаторы, интерфейс которых совпадает с интерфейсом некоторых служебных программ, требующих ввод конфиденциальной информации (паролей, криптографических ключей, номеров кредитных карточек);
- замаскированные закладки, которые маскируются под программные средства оптимизации работы компьютера (файловые архиваторы, дисковые дефрагментаторы) или под программы игрового и развлекательного назначения.

Чтобы программная закладка могла произвести какие-либо действия по отношению к другим программам или по отношению к данным, процессор должен приступить к исполнению команд, входящих в состав кода программной закладки. Это возможно только при одновременном соблюдении следующих условий:

- программная закладка должна попасть в оперативную память компьютера (если закладка относится к первому типу, то она должна быть загружена до начала работы другой программы, которая является целью воздействия закладки, или во время работы этой программы);
- работа закладки, находящейся в оперативной памяти, начинается при выполнении ряда условий, которые называются активизирующими.

Интересно, что иногда сам пользователь провоцируется на запуск исполняемого файла, содержащего код программной закладки. Известен такой случай. Среди пользователей свободно распространялся набор из архивированных файлов. Для извлечения файлов из него требовалось вызвать специальную утилиту, которая, как правило, есть почти у каждого пользователя и запускается после указания ее имени в командной строке. Однако мало кто из пользователей замечал, что в полученном наборе файлов уже имелась программа с таким же именем и что запускаюсь именно она. Кроме разархивирования файлов, эта программная закладка дополнительно производила ряд действий негативного характера.

С учетом замечания о том, что программная закладка должна быть обязательно загружена в оперативную память компьютера, можно выделить **резидентные закладки** (они находятся в оперативной памяти постоянно, начиная с некоторого момента и до окончания сеанса работы компьютера, т. е. до его перезагрузки или до выключения питания) **и нерезидентные** (такие закладки попадают в оперативную память компьютера аналогично резидентным, однако, в отличие от последних, выгружаются по истечении некоторого времени или при выполнении особых условий).

Существуют три основные группы деструктивных действий, которые могут осуществляться программными закладками:

- копирование информации пользователя компьютерной системы (паролей, криптографических ключей, кодов доступа, конфиденциальных электронных документов), находящейся в оперативной или внешней памяти этой системы либо в памяти другой компьютерной системы, подключенной к ней через локальную или глобальную компьютерную сеть;

- изменение алгоритмов функционирования системных, прикладных и служебных программ (например, внесение изменений в программу разграничения доступа может привести к тому, что она разрешит вход в систему всем без исключения пользователям вне зависимости от правильности введенного пароля);

- навязывание определенных режимов работы (например, блокирование записи на диск при удалении информации, при этом информация, которую требуется удалить, не уничтожается и может быть впоследствии скопирована хакером).

У всех программных закладок (независимо от метода их внедрения в компьютерную систему, срока их пребывания в оперативной памяти и назначения) имеется одна важная общая черта: они обязательно выполняют операцию записи в оперативную или внешнюю память системы. При отсутствии данной операции никакого негативного влияния программная закладка оказать не может. Ясно, что для целенаправленного воздействия она

должна выполнять и операцию чтения, иначе в ней может быть реализована только функция разрушения (например, удаление или замена информации в определенных секторах жесткого диска).

Троянской программой (троянцем, или троянским конем) называется:

- программа, которая, являясь частью другой программы с известными пользователю функциями, способна втайне от него выполнять некоторые дополнительные действия с целью причинения ему определенного ущерба;

- программа с известными ее пользователю функциями, в которую были внесены изменения, чтобы, помимо этих функций, она могла втайне от него выполнять некоторые другие (разрушительные) действия.

Таким образом, троянская программа — это особая разновидность программной закладки. Она дополнительно наделена функциями, о существовании которых пользователь даже не подозревает. Когда троянская программа выполняет эти функции, компьютерной системе наносится определенный ущерб. Однако то, что при одних обстоятельствах причиняет непоправимый вред, при других— может оказаться вполне полезным. К примеру, программу, которая форматирует жесткий диск, нельзя назвать троянской, если она как раз и предназначена для его форматирования (как это делает команда `format` операционной системы DOS). Но если пользователь, выполняя некоторую программу, совершенно не ждет, что она отформатирует его винчестер, — это и есть самый настоящий троянец.

Троянской можно считать любую программу, которая втайне от пользователя выполняет какие-то нежелательные для него действия. Эти действия могут быть любыми — от определения регистрационных номеров программного обеспечения, установленного на компьютере, до составления списка каталогов на его жестком диске. А сама троянская программа может маскироваться под текстовый редактор, под сетевую утилиту или любую программу, которую пользователь пожелает установить на свой компьютер.

Большинство троянских программ предназначено для сбора конфиденциальной информации. Их задача, чаще всего, состоит в выполнении действий, позволяющих получить доступ к данным, которые не подлежат широкой огласке. К таким данным относятся пользовательские пароли, регистрационные номера программ, сведения о банковских счетах и т. д. Остальные троянцы создаются для причинения прямого ущерба компьютерной системе, приводя ее в неработоспособное состояние.

Троянцы, входящие в распространенные компьютерные приложения, утилиты и операционные системы, представляют значительно большую угрозу компьютерам, на которых они установлены. поскольку их действия носят не деструктивный характер, а имеют целью сбор конфиденциальной информации о системе. Обнаружить такие троянские программы удастся, как правило, чисто случайно. А поскольку программное обеспечение, частью которого они являются, в большинстве случаев используется не только какой-то одной компанией, закупившей это программное обеспечение, но также на крупных Internet-серверах и, кроме того, распространяется через Internet, последствия могут оказаться самыми плачевными.

В настоящее время троянские программы можно отыскать практически где угодно. Они написаны для всех без исключения операционных систем и для любых платформ. Не считая случаев, когда троянские программы пишутся самими разработчиками программного обеспечения, троянцы распространяются тем же способом, что и компьютерные вирусы. Поэтому самыми подозрительными на предмет присутствия в них троянцев, в первую очередь, являются бесплатные и условно-бесплатные программы, скачанные из Internet, а также программное обеспечение, распространяемое на пиратских компакт-дисках.

Таким образом, троянские программы встречаются довольно часто и, следовательно, представляют серьезную угрозу безопасности компьютерных систем. Даже после того как троянская программа обнаружена, ее вредоносное влияние на компьютерную систему может ощущаться еще в

течение очень длительного времени. Ведь зачастую никто не может с уверенностью сказать, насколько сильно пострадала компьютерная система в результате проникновения в нее троянской программы.

Большинство программных средств, предназначенных для защиты от троянских программ, в той или иной степени использует так называемое согласование объектов. При этом в качестве объектов фигурируют файлы и каталоги, а согласование представляет собой способ ответить на вопрос, изменились ли файлы и каталоги с момента последней проверки. В ходе согласования характеристики объектов сравниваются с характеристиками, которыми они обладали раньше. Берется, к примеру, архивная копия системного файла и ее атрибуты сравниваются с атрибутами этого файла, который в настоящий момент находится на жестком диске. Если атрибуты различаются и никаких изменений в операционную систему не вносилось, значит в компьютер, скорее всего, проник троянец.

Одним из атрибутов любого файла является отметка о времени его последней модификации: всякий раз, когда файл открывается, изменяется и сохраняется на диске, автоматически вносятся соответствующие поправки. Однако отметка времени не может служить надежным индикатором наличия в системе троянца. Дело в том, что ею очень легко манипулировать. Можно подкрутить назад системные часы, внести изменения в файл, затем снова вернуть часы в исходное состояние, и отметка о времени модификации файла останется неизменной.

В борьбе с троянцами положиться на отметку о времени последней модификации файла и его размер нельзя, поскольку злоумышленник может их довольно легко подделать. Более надежной в этом отношении является так называемая контрольная сумма файла. Для ее подсчета элементы файла суммируются, и получившееся в результате число объявляется его контрольной суммой. Например, в операционной системе SunOS существует специальная утилита `sum`, которая выводит на устройство стандартного

вывода STDOUT контрольную сумму файлов, перечисленных в строке аргументов этой утилиты.

Однако и контрольную сумму в общем случае оказывается не так уж трудно подделать. ***Поэтому для проверки целостности файловой системы компьютера используется особая разновидность алгоритма вычисления контрольной суммы, называемая односторонним хэшированием.***

Функция хэширования называется односторонней, если задача отыскания двух аргументов, для которых ее значения совпадают, является труднорешаемой. Отсюда следует, что функция одностороннего хэширования может быть применена для того, чтобы отслеживать изменения, вносимые злоумышленником в файловую систему компьютера, поскольку попытка злоумышленника изменить какой-либо файл так, чтобы значение, полученное путем одностороннего хэширования этого файла, осталось неизменным, обречена на неудачу.

Клавиатурные шпионы

Одна из наиболее распространенных разновидностей программных закладок — клавиатурные шпионы. Такие программные закладки нацелены на перехват паролей пользователей операционной системы, а также на определение их легальных полномочий и прав доступа к компьютерным ресурсам.

Клавиатурные шпионы — явление отнюдь не новое в мире компьютеров. В свое время они разрабатывались и для OS/370, и для UNIX, и для DOS. Их поведение в общем случае является довольно традиционным: типовой клавиатурный шпион обманным путем завладевает пользовательскими паролями, а затем переписывает эти пароли туда, откуда их может без особого труда извлечь злоумышленник. Различия между клавиатурными шпионами касаются только способа, который применяется ими для перехвата пользовательских паролей. ***Соответственно все клавиатурные шпионы делятся на три типа — имитаторы, фильтры и заместители.***

Имитаторы

Клавиатурные шпионы этого типа работают по следующему алгоритму. Злоумышленник внедряет в операционную систему программный модуль, который имитирует приглашение пользователю зарегистрироваться для того, чтобы войти в систему. Затем внедренный модуль (в принятой терминологии — имитатор) переходит в режим ожидания ввода пользовательского идентификатора и пароля. После того как пользователь идентифицирует себя и введет свой пароль, имитатор сохраняет эти данные там, где они доступны злоумышленнику. Далее имитатор инициирует выход из системы (что в большинстве случаев можно сделать программным путем), и в результате перед глазами у ничего не подозревающего пользователя появляется сто одно, но на этот раз уже настоящее приглашение для входа в систему.

Обманутый пользователь, видя, что ему предлагается еще раз внести пароль, приходит к выводу о том, что он допустил какую-то ошибку во время предыдущего ввода пароля, и послушно повторяет всю процедуру входа в систему заново. Некоторые имитаторы для убедительности выдают на экран монитора правдоподобное сообщение о якобы совершенной пользователем ошибке. Например, такое: "НЕВЕРНЫЙ ПАРОЛЬ. ПОПРОБУЙТЕ ЕЩЕ РАЗ".

Перехват пароля зачастую облегчают сами разработчики операционных систем, которые не затрудняют себя созданием усложненных по форме приглашений пользователю зарегистрироваться для входа в систему. Подобное пренебрежительное отношение характерно для большинства версий операционной системы UNIX, в которых регистрационное приглашение состоит из двух текстовых строк, выдаваемых поочередно на экран терминала:

login:

password:

Системный процесс WinLogon, отвечающий в операционной системе Windows за аутентификацию пользователей, имеет свой собственный рабочий стол — совокупность окон, одновременно видимых на экране дисплея. Этот рабочий стол называется столом аутентификации. Никакой другой процесс, в том числе и имитатор, не имеет доступа к рабочему столу аутентификации и не может расположить на нем свое окно.

После запуска Windows на экране компьютера возникает так называемое начальное окно рабочего стола аутентификации, содержащее указание нажать на клавиатуре клавиши <Ctrl>+<Alt>+. Сообщение о нажатии этих клавиш передается только системному процессу WinLogon, а для остальных процессов, в частности, для всех прикладных программ, их нажатие происходит совершенно незаметно. Далее производится переключение на другое, так называемое регистрационное окно рабочего стола аутентификации. В нем-то как раз и размещается приглашение пользователю ввести свое идентификационное имя и пароль, которые будут восприняты и проверены процессом WinLogon.

Для перехвата пользовательского пароля внедренный в Windows имитатор обязательно должен уметь обрабатывать нажатие пользователем клавиш <Ctrl>+<Alt>+. В противном случае произойдет переключение на регистрационное окно рабочего стола аутентификации, имитатор станет неактивным и не сможет ничего перехватить, поскольку все символы пароля, введенные пользователем, минуют имитатор и станут достоянием исключительно системного процесса WinLogon. Как уже говорилось, процедура регистрации в Windows устроена таким образом, что нажатие клавиш <Ctrl>+<Alt>+ проходит бесследно для всех процессов, кроме WinLogon, и поэтому пользовательский пароль поступит именно ему.

Конечно, имитатор может попытаться воспроизвести не начальное окно рабочего стола аутентификации (в котором высвечивается указание пользователю одновременно нажать клавиши <Ctrl>+<Alt>+), а

регистрационное (где содержится приглашение ввести идентификационное имя и пароль пользователя). Однако при отсутствии имитаторов в системе регистрационное окно автоматически заменяется на начальное по прошествии короткого промежутка времени (в зависимости от версии Windows он может продолжаться от 30 с до 1 мин), если в течение этого промежутка пользователь не предпринимает никаких попыток зарегистрироваться в системе. Таким образом, сам факт слишком долгого присутствия на экране регистрационного окна должен насторожить пользователя Windows и заставить его тщательно проверить свою компьютерную систему на предмет наличия в ней программных закладок.

Степень защищенности Windows от имитаторов достаточно высока. Рассмотрение защитных механизмов, реализованных в этой операционной системе, позволяет сформулировать два необходимых условия, соблюдение которых является обязательным для обеспечения надежной защиты от имитаторов:

- системный процесс, который при входе пользователя в систему получает от него соответствующие регистрационное имя и пароль, должен иметь свой собственный рабочий стол, недоступный другим процессам;
- переключение на регистрационное окно рабочего стола аутентификации должно происходить абсолютно незаметно для прикладных программ, которые к тому же никак не могут повлиять на это переключение (например, запретить его).

Фильтры

Фильтры "охотятся" за всеми данными, которые пользователь операционной системы вводит с клавиатуры компьютера. Самые элементарные фильтры просто сбрасывают перехваченный клавиатурный ввод на жесткий диск или в какое-то другое место, к которому имеет доступ злоумышленник. Более, изощренные программные закладки этого типа

подвергают перехваченные данные анализу и отфильтровывают информацию, имеющую отношение к пользовательским паролям.

Фильтры являются резидентными программами, перехватывающими одно или несколько прерываний, которые связаны с обработкой сигналов от клавиатуры. Эти прерывания возвращают информацию о нажатой клавише и введенном символе, которая анализируется фильтрами на предмет выявления данных, имеющих отношение к паролю пользователя.

В общем случае можно утверждать, что если в операционной системе разрешается переключать клавиатурную раскладку во время ввода пароля, то для этой операционной системы возможно создание фильтра. Поэтому, чтобы обезопасить ее от фильтров, необходимо обеспечить выполнение следующих трех условий:

- во время ввода пароля переключение раскладок клавиатуры не разрешается ;*
- конфигурировать цепочку программных модулей, участвующих в работе с паролем пользователя, может только системный администратор;*
- доступ к файлам этих модулей имеет исключительно системный администратор.*

Заместители

Заместители полностью или частично подменяют собой программные модули операционной системы, отвечающие за аутентификацию пользователей. Подобного рода клавиатурные шпионы могут быть созданы для работы в среде практически любой многопользовательской операционной системы. Трудоемкость написания заместителя определяется сложностью алгоритмов, реализуемых подсистемой аутентификации, и интерфейсов между ее отдельными модулями. Также при оценке трудоемкости следует принимать во внимание степень документированности

этой подсистемы. В целом можно сказать, что задача создания заместителя значительно сложнее задачи написания имитатора или фильтра. Поэтому фактов использования подобного рода программных закладок злоумышленниками пока отмечено не было. Однако в связи с тем, что в настоящее время все большее распространение получает операционная система Windows, имеющая мощные средства защиты от имитаторов и фильтров, в самом скором будущем от хакеров следует ожидать более активного использования заместителей в целях получения несанкционированного доступа к компьютерным системам.

Поскольку заместители берут на себя выполнение функций подсистемы аутентификации, перед тем как приступить к перехвату пользовательских паролей они должны выполнить следующие действия:

- подобно компьютерному вирусу внедриться в один или несколько системных файлов;*
- использовать интерфейсные связи между программными модулями подсистемы аутентификации для встраивания себя в цепочку обработки введенного пользователем пароля.*

Для того чтобы защитить систему от внедрения заместителя, ее администраторы должны строго соблюдать адекватную политику безопасности. И что особенно важно, подсистема аутентификации должна быть одним из самых защищенных элементов операционной системы. Однако, как показываем практика, администраторы, подобно всем людям, склонны к совершению ошибок. А следовательно, соблюдение адекватной политики безопасности в течение неограниченного периода времени является невыполнимой задачей. Кроме того, как только заместитель попал в компьютерную систему, любые меры защиты от внедрения программных закладок перестают быть адекватными, и поэтому необходимо предусмотреть возможность использования эффективных средств обнаружения и удаления внедренных клавиатурных шпионов. Это

значит, что администратор должен вести самый тщательный контроль целостности исполняемых системных файлов и интерфейсных функций, используемых подсистемой аутентификации для решения своих задач.

Но и эти меры могут оказаться недостаточно эффективными. Ведь машинный код заместителя выполняется в контексте операционной системы, и поэтому заместитель может предпринимать особые меры, чтобы максимально затруднить собственное обнаружение. Например, он может перехватывать системные вызовы, используемые администратором для выявления программных закладок, с целью подмены возвращаемой ими информации. Или фильтровать сообщения, регистрируемые подсистемой аудита, чтобы отсеивать те, которые свидетельствуют о его присутствии в компьютере.

Логические «бомбы»

Логическая «бомба» [Logic bomb] - программа, которая запускается при определенных временных или информационных условиях для осуществления несанкционированного доступа к информации.

В отличие от вирусов, логические бомбы не делают своих копий.

Логическая бомба может быть «заложена» внутрь вредоносных программ другого типа, вызывая их срабатывание в заданное время. Кроме того, логическая бомба может находиться и внутри обычных программ, вызывая прекращение их работы в заданное время.

Логическая бомба может быть «доставлена» адресату при помощи электронной почты, вместе с вирусом или троянской программно. Логическая бомба может быть внедрена в программу или систему еще на этапе ее разработки.

Название отражает тот факт, что вредоносные действия выполняются не сразу после проникновения логической бомбы на компьютер, а через некоторое время или при выполнении некоторых условий.

После внедрения на компьютер логическая бомба может никак себя не проявлять в течение длительного времени. Как правило, логические бомбы начинают свое вредоносное действие тогда, когда это может привести к максимальному ущербу для компьютерной системы.

Логические бомбы (Logic bombs) — вид троянского коня - скрытые модули, встроенные в ранее разработанную и широко используемую программу. Являются средством компьютерного саботажа. Такой модуль является безвредным до определенного события, при наступлении которого он срабатывает (нажатие пользователем определенных кнопок клавиатуры, изменение в файле или наступление определенной даты или времени).

Логическая бомба является на сегодняшний день одной из самых распространенных хакерских атак. В самом деле, она достаточно легко реализуема, трудно обнаруживаема и может принести достаточно серьезный вред вашему компьютеру.

Взрыв логической бомбы может заключаться в форматировании жесткого диска, удалении файлов в случайном порядке, осуществлении сложно обнаруживаемых изменений в ключевых программах или шифровании важных файлов.

«Потайные двери»

Еще один способ создания дыры в системе безопасности изнутри называется «потайной дверью». Для этого в систему системным программистом внедряется специальная программа, позволяющая обойти нормальную процедуру проверки. Например, программист может добавить к программе регистрации кусок программы, пропускающий в систему пользователя с именем "zzzzz", независимо от того, что содержится в файле паролей.

Чтобы не допустить установки «потайных дверей» в систему, компании могут, например, ввести регулярные просмотры программ. При этом, когда программист закончил писать и тестировать программный модуль, модуль проверяется и помещается в базу данных. Периодически все программисты

команды собираются вместе и каждый из них поочередно разъясняет остальным, что делает его программа, строка за строкой. При этом вероятность обнаружения потайных дверей значительно увеличивается.

Задание к лабораторной работе.

Изучить принципы работы программных закладок, логических «бомб» и «троянских коней». Провести анализ возможностей программ и представить модели внедрения разных видов программных закладок в компьютерную систему (уровень операционной системы, программного обеспечения). Провести анализ средств защиты от перечисленных программ.

Изучение возможности использования логических «бомб» с целью защиты от незаконного использования авторских программ, путем дописывания дополнительных модулей в свою программу (например, курсовая работе по операционным системам), реализующих «потайные двери» и логическую «бомбу» без использования деструктивных действий, но с возможной модификацией структур, с целью защиты своей авторской программы от незаконного использования.

Вопросы к лабораторной работе:

- 1) К каким видам угроз безопасности относятся программные закладки?
- 2) Способы защиты от клавиатурных шпионов.
- 3) Какого типа программную закладку можно реализовать, не являясь разработчиком программных систем, но имея повышенные привилегии (например, администратор системы)
- 4) Привести пример использования «потайных дверей» не в деструктивных целях.
- 5) Привести пример использования логических «бомб» не в деструктивных целях.

Лабораторная работа №2

Тема: Базовые алгоритмы шифрования. Использование симметричного алгоритма «Сеть Фейстеля» для шифрования файлов на диске.

Цель: Изучить базовые алгоритмы шифрования и написать программу, выполняющую шифрование текста с помощью одного из базовых алгоритмов, выполнив предварительно модификацию базового алгоритма шифрования. Изучить симметричный алгоритм шифрования «Сеть Фейстеля» и написать программу, выполняющую шифрование выбранных файлов на диске с помощью симметричного алгоритма «Сеть Фейстеля». На вход "Сети Фейстеля" подать предварительно зашифрованное сообщение базовым алгоритмом шифрования.

Методические указания к лабораторной работе

(часть 1. Базовые алгоритмы шифрования)

Рассматриваемые простейшие методы кодирования заключаются в видоизменении информации таким образом, чтобы при попытке ее прочесть, злоумышленник не увидел ничего, кроме бессмысленной последовательности символов, а при попытке подделать или переделать документ, читатель, увидев ту же бессмыслицу, сразу понял бы это. Это и называется шифрованием или криптографией.

Зашифровав текст, его можно переправлять любым доступным способом, но никто не сможет узнать содержимое послания. Хотя, последнее не совсем верно, так как злоумышленник может попытаться раскрыть код.

Наука о раскрытии шифров – криптоанализ – всегда развивалась параллельно шифрованию. В последнее время, обе науки начали опираться на серьезный математический анализ, что только усилило их противостояние.

Не всегда криптоанализ используется злоумышленником. Дело в том, что преступная организация также может зашифровать послание. Тогда,

дешифрация сообщения может оказаться жизненно важной. Поэтому, употребляя слово злоумышленник, говоря о шифровании, не совсем корректно, тем более, что в военном конфликте понятия прав и виноват весьма и весьма относительны. Но если сообщение шифруется, всегда есть опасность попытки его дешифрации.

Именно поэтому появился термин криптостойкость, то есть устойчивость алгоритма к криптоанализу.

Прежде чем говорить о более сложных алгоритмах шифрования, рассмотрим базовые алгоритмы, которые частично используются в современных алгоритмах шифрования. К простейшим алгоритмам шифрования относятся:

- шифр Полибия;
- шифр Цезаря;
- шифр Ришеля;
- перестановочный шифр;
- шифр Виженера;
- шифр Плейфера;
- шифрование с ключом;
- шифр Вернама;
- шифр «Люцифер»;

Многие криптосистемы, как простые, так и сложные основываются на принципах подстановки, перестановки и гаммирования. Прежде чем рассматривать простейшие алгоритмы шифрования, опишем в чем суть подстановок, перестановок и гаммирования, так как эти принципы являются базовыми.

Перестановки.

Перестановкой σ набора целых чисел $(0, 1, \dots, N-1)$ называется его переупорядочение. Для того чтобы показать, что целое i перемещено из позиции i в позицию $\sigma(i)$, где $0 \leq (i) < n$, будем использовать запись

$$\sigma=(\sigma(0), \sigma(1),..., \sigma(N-1)).$$

Число перестановок из $(0,1,...,N-1)$ равно $n!=1*2*...*(N-1)*N$. Введем обозначение σ для взаимно-однозначного отображения (гомоморфизма) набора $S=\{s_0,s_1, ...,s_{N-1}\}$, состоящего из n элементов, на себя.

$$\sigma: S \rightarrow S$$

$$\sigma: s_i \rightarrow s_{\sigma(i)}, 0 \leq i < n$$

Будем говорить, что в этом смысле σ является *перестановкой элементов* S . И, наоборот, автоморфизм S соответствует перестановке целых чисел $(0,1,2,..., n-1)$.

Криптографическим преобразованием T для алфавита Z_m называется последовательность автоморфизмов: $T=\{T^{(n)}; 1 \leq n < \infty\}$

$$T^{(n)}: Z_{m,n} \rightarrow Z_{m,n}, 1 \leq n < \infty$$

Каждое $T^{(n)}$ является, таким образом, перестановкой n -грамм из $Z_{m,n}$.

Поскольку $T^{(i)}$ и $T^{(j)}$ могут быть определены независимо при $i \neq j$, число криптографических преобразований исходного текста размерности n равно $(m^n)!$ (Здесь и далее m - объем используемого алфавита.). Оно возрастает непропорционально при увеличении m и n : так, при $m=33$ и $n=2$ число различных криптографических преобразований равно $1089!$. Отсюда следует, что потенциально существует большое число отображений исходного текста в шифрованный.

Практическая реализация криптографических систем требует, чтобы преобразования $\{T_k: k \in K\}$ были определены алгоритмами, зависящими от относительно небольшого числа параметров (ключей).

Системы подстановок.

Определение Подстановкой π на алфавите Z_m называется автоморфизм Z_m , при котором буквы исходного текста t замещены буквами шифрованного текста $\pi(t)$:

$$Z_m \rightarrow Z_m; \pi: t \rightarrow \pi(t).$$

Набор всех подстановок называется симметрической группой Z_m и будет в дальнейшем обозначаться как $SYM(Z_m)$.

Утверждение $SYM(Z_m)$ с операцией произведения является группой, т.е. операцией, обладающей следующими свойствами:

Замкнутость: произведение подстановок $\pi_1\pi_2$ является подстановкой:

$$\pi: t \rightarrow \pi_1(\pi_2(t)).$$

Ассоциативность: результат произведения $\pi_1\pi_2\pi_3$ не зависит от порядка расстановки скобок:

$$(\pi_1\pi_2)\pi_3 = \pi_1(\pi_2\pi_3)$$

Существование нейтрального элемента: подстановка i , определяемая как $i(t)=t$, $0 \leq t < m$, является нейтральным элементом $SYM(Z_m)$ по операции умножения: $i\pi = \pi i$ для $\forall \pi \in SYM(Z_m)$.

Существование обратного: для любой подстановки π существует единственная обратная подстановка π^{-1} , удовлетворяющая условию

$$\pi\pi^{-1} = \pi^{-1}\pi = i.$$

Число возможных подстановок в симметрической группе Z_m называется *порядком* $SYM(Z_m)$ и равно $m!$.

Определение. *Ключом* подстановки k для Z_m называется последовательность элементов симметрической группы Z_m :

$$k = (p_0, p_1, \dots, p_{n-1}, \dots), p_n \in SYM(Z_m), 0 \leq n < \infty$$

Подстановка, определяемая ключом k , является криптографическим преобразованием T_k , при помощи которого осуществляется преобразование

n -граммы исходного текста $(x_0, x_1, \dots, x_{n-1})$ в n -грамму шифрованного текста $(y_0, y_1, \dots, y_{n-1})$:

$$y_i = p(x_i), \quad 0 \leq i < n$$

где n – произвольное ($n=1,2,\dots$). T_k называется моноалфавитной подстановкой, если p неизменно при любом i , $i=0,1,\dots$, в противном случае T_k называется многоалфавитной подстановкой.

Примечание. К наиболее существенным особенностям подстановки T_k относятся следующие:

1. *Исходный текст шифруется посимвольно.* Шифрования n -граммы $(x_0, x_1, \dots, x_{n-1})$ и ее префикса $(x_0, x_1, \dots, x_{s-1})$ связаны соотношениями

$$T_k(x_0, x_1, \dots, x_{n-1}) = (y_0, y_1, \dots, y_{n-1})$$

$$T_k(x_0, x_1, \dots, x_{s-1}) = (y_0, y_1, \dots, y_{s-1})$$

2. *Буква шифрованного текста y_i является функцией только i -й компоненты ключа p_i и i -й буквы исходного текста x_i .*

В потоковых шифрах, т. е. при шифровании потока данных, каждый бит исходной информации шифруется независимо от других с помощью гаммирования.

Гаммирование – наложение на открытые данные гаммы шифра (случайной или псевдослучайной последовательности единиц и нулей) по определенному правилу. Обычно используется "исключающее ИЛИ", называемое также сложением по модулю 2 и реализуемое в ассемблерных программах командой XOR. Для расшифровывания та же гамма накладывается на зашифрованные данные.

При однократном использовании случайной гаммы одинакового размера с зашифровываемыми данными взлом кода невозможен (так называемые криптосистемы с одноразовым или бесконечным ключом). В данном случае "бесконечный" означает, что гамма не повторяется.

В некоторых потоковых шифрах ключ короче сообщения. Так, в системе Вернама для телеграфа используется бумажное кольцо, содержащее гамму. Конечно, стойкость такого шифра не идеальна.

Понятно, что обмен ключами размером с шифруемую информацию не всегда уместен. Поэтому чаще используют гамму, получаемую с помощью генератора псевдослучайных чисел (ПСЧ). В этом случае ключ - порождающее число (начальное значение, вектор инициализации, *initializing value*, IV) для запуска генератора ПСЧ. Каждый генератор ПСЧ имеет период, после которого генерируемая последовательность повторяется. Очевидно, что период псевдослучайной гаммы должен превышать длину шифруемой информации.

Генератор ПСЧ считается корректным, если наблюдение фрагментов его выхода не позволяет восстановить пропущенные части или всю последовательность при известном алгоритме, но неизвестном начальном значении.

При использовании генератора ПСЧ возможно несколько вариантов:

1. Побитовое шифрование потока данных. Цифровой ключ используется в качестве начального значения генератора ПСЧ, а выходной поток битов суммируется по модулю 2 с исходной информацией. В таких системах отсутствует свойство распространения ошибок.

2. Побитовое шифрование потока данных с обратной связью (ОС) по шифртексту. Такая система аналогична предыдущей, за исключением того, что шифртекст возвращается в качестве параметра в генератор ПСЧ. Характерно свойство распространения ошибок. Область распространения ошибки зависит от структуры генератора ПСЧ.

3. Побитовое шифрование потока данных с ОС по исходному тексту. Базой генератора ПСЧ является исходная информация. Характерно свойство неограниченного распространения ошибки.

4. Побитовое шифрование потока данных с ОС по шифртексту и по исходному тексту.

Чтобы получить линейные последовательности элементов гаммы, длина которых превышает размер шифруемых данных, используются *датчики ПСЧ*. На основе теории групп было разработано несколько типов таких датчиков.

Конгруэнтные датчики

В настоящее время наиболее доступными и эффективными являются *конгруэнтные* генераторы ПСП. Для этого класса генераторов можно сделать математически строгое заключение о том, какими свойствами обладают выходные сигналы этих генераторов с точки зрения периодичности и случайности.

Одним из хороших конгруэнтных генераторов является линейный конгруэнтный датчик ПСЧ. Он вырабатывает последовательности псевдослучайных чисел $T(i)$, описываемые соотношением

$$T(i+1) = (A \cdot T(i) + C) \bmod m,$$

где A и C - константы, $T(0)$ - исходная величина, выбранная в качестве порождающего числа. Очевидно, что эти три величины и образуют ключ.

Такой датчик ПСЧ генерирует псевдослучайные числа с определенным периодом повторения, зависящим от выбранных значений A и C . Значение m обычно устанавливается равным 2^n , где n - длина машинного слова в битах. Датчик имеет максимальный период M до того, как генерируемая последовательность начнет повторяться. По причине, отмеченной ранее, необходимо выбирать числа A и C такие, чтобы период M был максимальным. Как показано Д. Кнудом, линейный конгруэнтный датчик ПСЧ имеет максимальную длину M тогда и только тогда, когда C - нечетное, и $A \bmod 4 = 1$.

Для шифрования данных с помощью датчика ПСЧ может быть выбран ключ любого размера. Например, пусть ключ состоит из набора чисел $x(j)$ размерностью b , где $j=1, 2, \dots, n$. Тогда создаваемую гамму шифра G можно представить как объединение непересекающихся множеств $H(j)$.

Шифрование с помощью датчика ПСЧ является довольно распространенным криптографическим методом. Во многом качество шифра, построен-

ного на основе датчика ПСЧ, определяется не только и не столько характеристиками датчика, сколько алгоритмом получения гаммы. Один из фундаментальных принципов криптологической практики гласит, даже сложные шифры могут быть очень чувствительны к простым воздействиям.

Рассмотрим базовые алгоритмы подробнее.

Шифр Полибия.

Самый, пожалуй, древний из всех известных, хотя и не такой распространенный, как шифр Цезаря. Полибий, греческий историк, умер за тридцать лет до появления Цезаря. Суть его метода в том, что составляется прямоугольник (Доска Полибия), например такой, как представлен на рисунке 1.

	А	Б	В	Г	Д	Е
А	А	Б	В	Г	Д	Е
Б	Ж	З	И	Й	К	Л
В	М	Н	О	П	Р	С
Г	Т	У	Ф	Х	Ц	Ч
Д	Ш	Щ	Ъ	Ы	Ь	Э
Е	Ю	Я	.	,	-	

Рисунок 1– Доска Полибия

Каждая буква может быть представлена парой букв, указывающих строку и столбец, в которых расположена данная буква. Так, представлением букв П, О будут ВГ, ВВ соответственно, а сообщение ШИФР ПОЛИБИЯ зашифруется как ДАБВГВВДЕЕВГВВБЕБВАББВЕБ. Хотя эта система и является более древней, но устойчивость к раскрытию у нее гораздо выше, несмотря на то, что наблюдается увеличение количества символов зашифрованного текста.

Шифр Цезаря.

Первый документально подтвержденный шифр: Гай Юлий Цезарь, желая защитить свои записи, заменял каждую букву следующей по алфавиту. Можно расширить этот алгоритм, поставив в соответствие каждой букве, какую либо другую букву, разумеется, без повторений. Можно даже заменять

буквы совершенно сторонними символами, это не повлияет на криптостойкость.

Во времена Цезаря, когда мало кто умел читать, да и статистика не была особенно развитой, такой шифр было почти невозможно взломать. Но теперь этот способ шифрования используют только в качестве головоломок.

Объяснить это легко. Посчитав вхождение каждого символа в тексте, и зная статистические особенности языка, то есть частоту появления каждого символа в текстах, можно смело заменить часть символов на настоящие. Остальной текст можно получить исходя из избыточности текстов на естественных языках. Очевидно также, что шифр Цезаря можно использовать только для текстов.

Это один из самых старых шифров, имеющий ряд серьезных недостатков. Огромное семейство подстановок Цезаря названо по имени римского императора

Гая Юлия Цезаря, который поручал Марку Туллию Цицерону составлять послания с использованием 50-буквенного греческого алфавита со сдвигом в 3 знака и дальнейшей подстановкой. Информация об этом дошла к нам от Светония. Попробуем воспроизвести этот метод шифрования для русского алфавита. Для этого создадим таблицу соответствия (рис. 2):

ю	а	б	в	г	д	е	ф	х	й	к	л	м	н	о	п	я	р	с	т	у	ж	в	ь	ы	э	ш	з	щ	ч	ъ
ц	д	е	ф	г	х	й	к	л	м	н	о	п	я	р	с	т	у	ж	в	ь	ы	э	ш	з	щ	ч	ъ	ю	а	б

Рисунок 2— Пример таблицы соответствия букв русского алфавита

Шифр Виженера.

Совершенно “естественно” выглядит криптотекст, полученный с помощью криптосистемы Виженера. Соответственно, защищенность этого алгоритма гораздо выше предыдущих. В основе лежит “замешивание” по определенным законам исходного текста в “мусоре”, или посторонних символах. Зашифрованный текст получается гораздо длиннее исходного, причем эта разница и определяет реальную криптостойкость системы. Суть

метода Ришелье в следующем. Создается подобная решетка с прорезями вместо крестов (рис. 3):

X	X						X		X
X	X				X			X	
						X			
X		X	X	X					X
							X		X
	X			X	X		X		X

Рисунок 3 – Пример решетки Ришелье

Далее в прорезях отверстий пишется текст, решетка снимается и все оставшееся пространство заполняется “мусором”, затем для усложнения расшифровки криптотекст можно выписать в ряд. Для длинных сообщений этот метод применяется несколько раз. Кстати говоря, выходной криптотекст можно оформить с помощью “мусора” и в виде смыслового послания. Выполнить обратное преобразование в исходный текст можно, зная размер блока и имея аналогичную решетку.

Перестановочные шифры.

Этот шифр тоже является одним из первых, он предполагает разбиение текста на блоки, а затем перемешивание символов внутри блока. Это можно сделать разными методами. В первых вариантах этого шифра использовалась таблица, которую заполняли по строкам исходными данными, а зашифрованные данные читали по столбцам.

Криптостойкость этого шифра также оставляет желать лучшего. Дело в том, что филологи уже давно установили, что для прочтения текста не важен порядок букв в слове, достаточно чтобы на месте были первая и последняя буквы слова. Конечно, размер блока не может совпадать с размерами всех слов текста, то есть перемешивание будет более серьезным, но научиться читать такой текст не очень сложно.

Рассмотрим этот метод на несложном алгоритме простой столбцевой перестановки. Предварительно условимся, что число столбцов равно пяти.

Допустим, имеется текст, который требуется зашифровать: “ПРОСТЫЕ ПЕРЕСТАНОВКИ ВНОСЯТ СУМЯТИЦУ”. Запишем этот текст слева направо, сверху вниз, заполняя последовательно пять столбцов (рис 4):

1	2	3	4	5
П	Р	О	С	Т
Ы	Е		П	Е
Р	Е	С	Т	А
Н	О	В	К	И
	В	Н	О	С
Я	Т		С	У
М	Я	Т	И	Ц
У				

Рисунок 4 – Пример простой столбцовой перестановки

Теперь зададимся ключом, например 35142, и выпишем в ряд столбцы, начиная с третьего, в соответствии с ключом: “О СВН Т ТЕАИСУЦ ПЫРН ЯМУСПТКОСИ РЕЕОВТЯ”. Получилось довольно неплохо. Для расшифровки требуется количество символов зашифрованного текста разделить на 5 и сформировать столбцы в соответствии с ключом.

В итоге получится исходная таблица, из которой можно прочесть текст построчно. Криптостойкость увеличится, если к полученному зашифрованному тексту применить повторное шифрование с другим размером таблицы и ключом или с перестановками не столбцов, а рядов. Перестановочные шифры очень хорошо сочетать с другими, таким образом повышая криптостойкость.

Шифр Виженера.

Наиболее известной и по праву одной из старейших многоалфавитных (одному шифруемому символу соответствует более одного символа) криптосистем является система известного французского криптографа графа Блейза Виженера (1523-1596).

Для преобразования строится алфавитный квадрат с построчным сдвигом символов в каждом последующем ряду (рисунок 5)

АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ
БВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯА
ВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБ
ГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВ
ДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГ
ЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГД
ЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕ
ЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖ
ИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗ
ЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИ
КЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙ
ЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙК
МНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛ
НОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМ
ОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМН
ПРСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНО
РСТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОП
СТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПР
ТУФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРС
УФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТ
ФХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТУ
ХЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТУФ
ЦЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТУФХ
ЧШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТУФХЦ
ШЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧ
ЩЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШ
ЪЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩ
ЫЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪ
ЬЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫ
ЭЮЯАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬ
ЮЯАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭ
ЯАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮ

Рисунок 5 – Квадрат Виженера

Далее выбирается ключ. Затем пишется шифруемая фраза, а под ней циклически записывается ключ. Преобразование производится так: шифруемому символу соответствует символ, находящийся на пересечении буквы ключа (столбец) и буквы исходного текста (строка).

Одним и тем же буквам исходного текста соответствуют не всегда одинаковые символы, то есть символы могут принимать различные значения, что является несомненным плюсом в плане криптостойкости зашифрованного текста.

Далее выбирается ключ, например “ГРАФДРАКУЛА”. Затем пишется шифруемая фраза, а под ней циклически записывается ключ. Для примера возьмем фразу “ОТРЯД ЖДЕТ УКАЗАНИЙ” и исключим из нее пробелы: “ОТРЯДЖДЕТУКАЗАНИЙ” – исходный текст, “ГРАФДРАКУЛАГРАФДР” – циклический ключ. Преобразование производится так: шифруемому символу соответствует символ, находящийся на пересечении буквы ключа (столбец) и буквы исходного текста (строка).

Получим: “СВРУИЦДПЕЮКГЧАБМЩ”. Как видно, одним и тем же буквам исходного текста соответствуют не всегда одинаковые символы, то

есть символы могут принимать различные значения, что является несомненным плюсом в плане криптостойкости зашифрованного текста.

Квадрат может быть каким угодно, главное, чтобы он был легко воспроизводимым и левый столбец, так же как и верхний ряд, соответствовали всему алфавиту. В качестве альтернативного варианта это может быть известный квадрат Френсиса Бьюфорта, где строки – зеркальное отражение строк квадрата Виженера. Для усложнения строки могут быть переставлены местами в определенном порядке и т. д. Усложненной разновидностью этого метода является метод шифрования с автоключом, предложенный математиком Дж. Кардано (XVI в.). Суть его в том, что, кроме циклического ключа, есть еще один, записываемый однократно перед циклическим ключом – первичный ключ. Этот нехитрый прием придает дополнительную стойкость алгоритму.

Шифр Плейфейра.

Назван так в честь своего разработчика. Алгоритм заключается в следующем: сначала составляется равносторонний алфавитный квадрат. Для русского алфавита подойдет квадрат 5x6. Задается ключевая фраза без повторов букв, например “БАРОН ЛИС”. Фраза вписывается в квадрат без пробелов, далее последовательно вписываются недостающие буквы алфавита в правильной последовательности. Одну букву придется исключить для выполнения условия равносторонности. Пусть это будет Ъ. Примем допущение, что Ъ=Ь. Итак, получился квадрат, который легко восстановить по памяти:

Теперь условия для преобразования:

- 1) Текст должен иметь четное количество букв и делиться на биграммы (сочетания по две буквы), недостающую часть текста дополняем самостоятельно одним (любым) символом. Например, текст “ОСЕНЬ” преобразуется в “ОСЕНЬА”.

- 2) Биграмма не должна содержать одинаковых букв – “СОСНА” – “СО СН АБ”.

Б	А	Р	О	Н
Л	И	С	В	Г
Д	Е	Ж	З	К
М	П	Т	У	Ф
Х	Ц	Ч	Ш	Щ
Ь	Ы	Э	Ю	Я

Рисунок 6 – Пример алфавитного квадрата для шифрования методом Плейфера

Правила преобразования:

Если биграмма не попадает в одну строку или столбец, то мы смотрим на буквы в углах прямоугольника, образованного рассматриваемыми буквами. Например: “ДИ” = “ЛЕ”, “БУ” = “МЮ” и т. д.

Если биграмма попадает в одну строку (столбец), мы циклично смещаемся на одну букву вправо (вниз). Например, “ЛС”=“ИВ”, “ТЭ”=“ЧР”.

Итак, мы готовы преобразовать текст. Приступим к шифрованию: “СОВЕЩАНИЕ СОСТОИТСЯ В ПЯТНИЦУ” – первоначально преобразуется в биграммы: “СО ВЕ ЩА НИ ЕС ОС ТО ИТ СЯ ВП ЯТ НИ ЦУ”. Выполнив преобразование по вышеизложенным правилам, получим биграммы: “РВ ЗИ НЦ ГА ИЖ ВР РУ ПС ЭГ УИ ФЭ ГА ПШ”.

Обратите внимание на то, что одинаковые буквы на входе дали разные буквы на выходе криптосистемы, а одинаковые буквы на выходе совсем не соответствуют одинаковым на входе. Таким образом, криптотекст “РВЗИНЦГАИЖВРРУПСЭГУИ” выглядит гораздо более “симпатично”, чем после прямой замены моноалфавитной криптосистемы.

Правила преобразования могут значительно варьироваться. Алфавит может быть упрощен до логического минимума с целью сокращения квадрата. Например, можно переводить текст в транслитерацию и писать латинскими буквами, тогда квадрат будет размером 5x5.

Шифрование с ключом.

Этот метод относится к современному периоду.

Текст представляется в двоичном формате, создается двоичный код некоторого размера (ключ), двоичный текст разбивается на блоки того же размера и каждый блок суммируется по модулю два с ключом. Обратное преобразование выполняется тем же методом.

Фактически, это «современный шифр Цезаря», просто подстановка выполняется не посимвольно, а словами. Правда, понятие слова здесь несколько иное – это блок заданной длины.

Криптостойкость этого метода намного выше, чем у предыдущих шифров. Без использования специальных методов криптоанализа его будет очень сложно взломать. Но в настоящее время этот метод используют только для не очень важных документов, так как криптоанализ, в совокупности с современными вычислительными средствами, позволяет не только прочесть текст сообщения, но и во многих случаях получить ключ, что губительно для системы безопасности.

Система Вернама.

В 1918 году, Шеннон сформулировал и доказал свою «пессимистическую теорему»: шифр является абсолютно криптостойким тогда и только тогда, когда энтропия ключа больше либо равна энтропии исходного текста. Из теоремы следует, что длина ключа должна быть соразмерна с длиной текста. Если быть более точным, длина ключа должна равняться длине текста.

На основе этой теоремы построена система Вернама. Каждый бит ключа этой системы равновероятно равен 0 либо 1. Длина ключа равна длине текста. Ключ используется только один раз. Доказано, что этот шифр абсолютно не вскрываем. Не будем этого доказывать, но обоснуем.

Для вскрытия зашифрованного сообщения необходимо перебрать все возможные комбинации ключей (это единственный способ, так как ключ используется лишь единожды), и среди полученных значений текста выбрать исходный, пусть это будет текст на русском языке. Допустим, что

злоумышленник может это сделать мгновенно, то есть опустим временную сложность такой задачи. Для текста любой длины, будет много бессмысленных результатов и их можно отбросить, но злоумышленник получит все возможные тексты заданной длины, в том числе наш текст. Но ничто и никогда не укажет на то, какой из текстов является исходным.

Рассмотрим это на примере четырех буквенных имен. Пусть зашифровано имя «Даша», а ключ «АБВГ». Если используется кодировка ASCII, то зашифрованный текст выглядит следующим образом: ♦!j#. Как говорилось ранее, единственный способ дешифровать текст, полный перебор всех возможных ключей. Но очевидно, что существует такой ключ, суммирование с которым зашифрованного текста даст в результате «Маша», или даже «Миша». Но действительно, невозможно ответить на вопрос, какое из имен является исходным.

Возможно, что аналитик знает часть текста, например, он знает что это договор на продажу недвижимости, но не знает цену. В результате перебора, если на это хватит времени, он получит все возможные цены «заданной длины». То есть если цена трехзначная, аналитиком будет получена девятьсот одна различная цена. Таким образом, взлом системы Вернама не приведет к получению новой информации и является абсолютно невскрываемым.

Недостаток этой системы – длина ключа. Сам Вернам предлагал пересылать ключ курьером. При этом если курьер не довез ключ, можно сгенерировать новый ключ и снова его отправить. Однако такой механизм чрезвычайно неудобен, поэтому эта система не используется в настоящее время.

Этот шифр при его идеальности лишен помехоустойчивости, потому как изменение одного бита, практически невозможно отследить.

Шифры, основанные на системе Вернама.

Если мы не можем использовать очень длинный ключ, следует использовать короткий, удлиняя его по некоторому правилу. Было предпринято много попыток создать шифр, как можно более приближенный к системе Вернама.

Первой попыткой было использование ключа в качестве параметров генератора псевдослучайных последовательностей. На деле все системы сводились именно к этому. Но в некоторых случаях ключ получали за счет его преобразований, например, с помощью нелинейной функции.



Рисунок 7 - Генерация ключа, для имитации системы Вернама

Наиболее удачной была схема, в которой ключ подавался на вход блоку преобразований (рисунок 7) одновременно с выходом двоичного счетчика, подключенного к таймеру. Для передачи сообщения правда требовалась большая синхронизация, нежели для системы Вернама, но вполне приемлемая.

Шифр «Люцифера».

Этот шифр изначально разрабатывался для аппаратной реализации. Именно при его создании одновременно использовались новейшие достижения в математике и теории кодирования и опыт веков.

«Люцифер» был создан фирмой IBM и долго использовался для внутренних нужд, так как аппаратная реализация системы шифрования имеет некоторые недостатки.

Авторы шифра предложили совместить два основных подхода к шифрованию: подстановки и перестановки.

Сначала, текст разбивается на блоки по n бит. Затем они поступают на вход мультиплексора, который раскладывает блок по модулю 2^n . Биты перемешиваются, и обрабатываются демультиплексором, который собирает биты в блок длиной n бит.

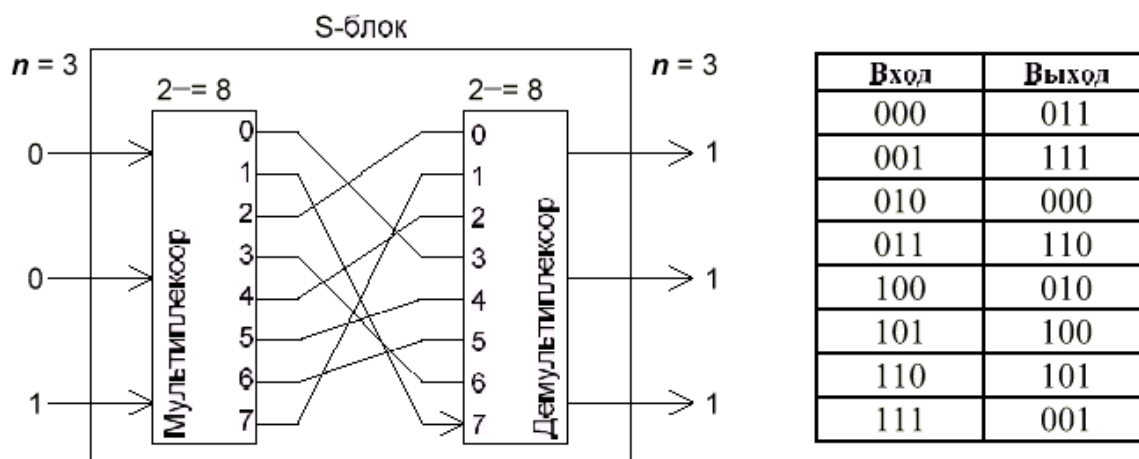


Рисунок 8 - Подстановки «Люцифера»

Преимущество таких постановок (Рисунок 8) заключается в нелинейности преобразований. По большому счету, можно любой входной поток превратить в любой выходной, но, разумеется, это производится на этапе разработки.

Выполнив подстановку, следует выполнить перестановки. Это делается соответствующей схемой, в которой просто перемешиваются биты (рисунок 9).

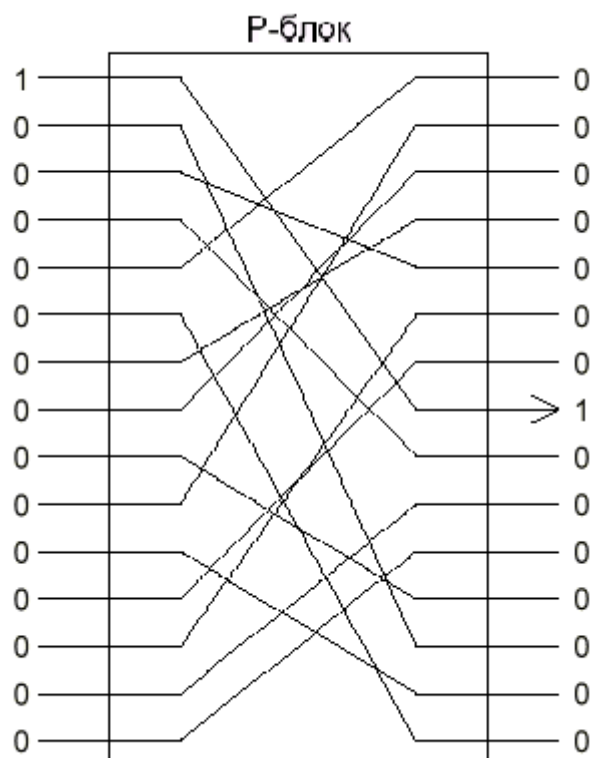


Рисунок 9 - Подстановки «Люцифера»

Узнать коммутацию такого блока довольно легко, достаточно подавать ровно одну единицу на вход, и получать ровно одну единицу на выходе. Однако, совместив оба подхода, и создавая как бы несколько итераций, можно очень сильно запутать аналитика (рисунок 10).

Легко увидеть, что если на вход подать одну единицу, на выходе получается больше единиц, чем нулей. Криптоаналитик, получивший текст, зашифрованный «Люцифером» будет поставлен в тупик.

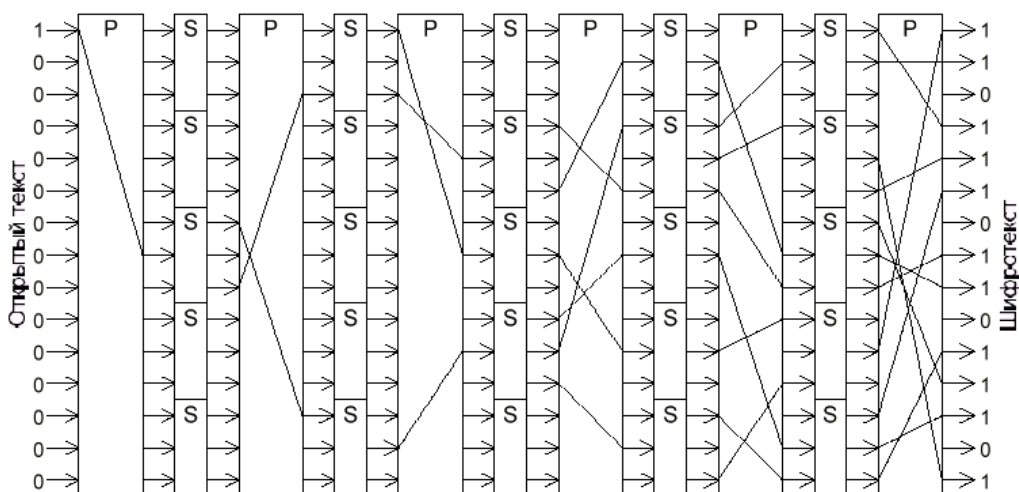


Рисунок 10 - «Люцифер»

Для этого шифра очень важно правильно подобрать параметры преобразований обоих типов. Дело в том, что при неудачном выборе параметров преобразований, они могут свестись к простой перестановке, которую легко раскрыть. Поэтому в аппаратную реализацию шифра встроили «сильные параметры».

Однако, аппаратная реализация шифра не позволяет его широкого применения, так как блок подстановок выполняет преобразование, схожее с суммированием с ключом, а суммирование с одним и тем же ключом ставит шифр под удар. С одной стороны, все, кто имеет доступ к системе, знают ключ, поэтому может произойти утечка, с другой стороны, аналитик, получивший достаточно много шифровок сможет так дешифровать сообщение.

Было несколько попыток избавиться от этого недостатка, но наиболее удачная из них заключалась в том, чтобы каждый блок подстановок заменить двумя, при этом для шифрования, пользователь указывает первый или второй вид подстановок выполнять, создавая двоичный ключ (рисунок 11).

Такая реализация шифра была гораздо более криптостойкой и использовалась некоторое время.

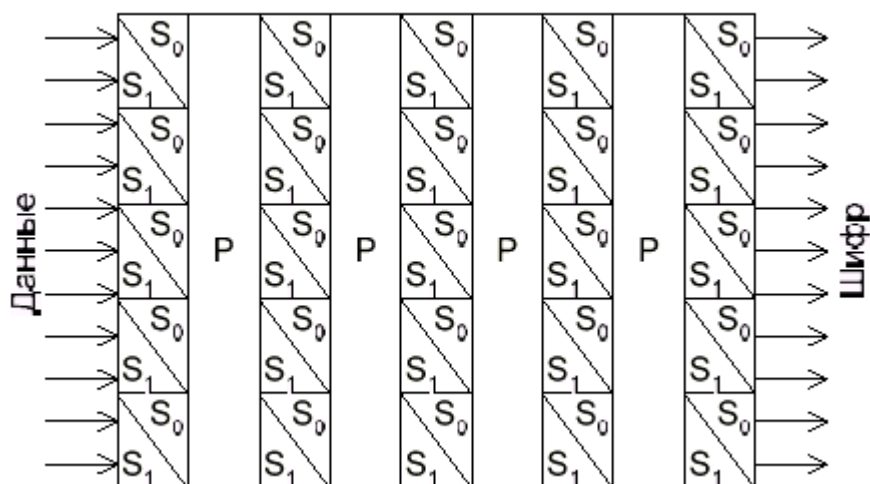


Рисунок 11 - «Люцифер» с ключом

Методические указания к лабораторной работе

(Часть 2. Сеть Фейстеля)

Действительно криптостойкий шифр на основе «Люцифера» был предложен в 1973 году Хорстом Фейстелем. Этот алгоритм на самом деле является общим случаем «Люцифера».

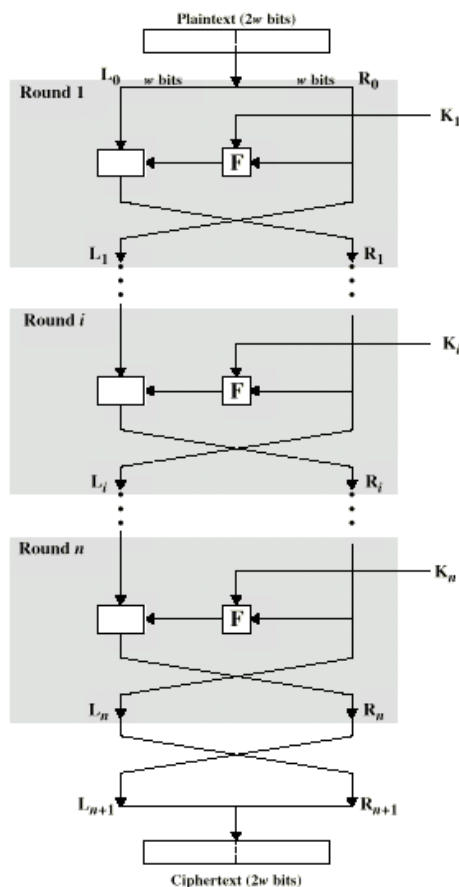


Рисунок 12 - Классическая сеть Фейстеля

Классическая (или простая) сеть Фейстеля (рисунок 12) разбивает исходный текст на блок длиной $2n$ бит, каждый блок разбивается на два потока L (левый) и R (правый). Далее выполняется преобразование по формуле:

$$\begin{aligned} L_{i+1} &= R_i \\ R_{i+1} &= L_i \oplus F(R_i, K_i) \end{aligned}$$

где K_i – ключ, F_i – функция преобразования.

Каждое такое действие называется раундом. Обычно используется 8-16 раундов. При этом, ключ в каждом раунде может быть как частью общего ключа, так и ее модификацией по некоторому правилу: следует отметить, что это в значительной мере влияет на криптостойкость. Также важным является правило, по которому текст разбивается на потоки.

Ключ, с которым суммируется текст, не повторяется из блока в блок, а изменяется в зависимости от самого текста, это значительно увеличивает его энтропию, и, следовательно, увеличивает криптостойкость метода.

Процедура расшифровки очевидна, следует применить те же формулы, но в обратном порядке. Некоторые исследователи нашли способ применять один и тот же алгоритм и для шифровки, и для расшифровки, для этого необходимо удвоить число раундов, и во второй половине расположить функции и ключи в обратном порядке. Но как показали эксперименты, криптостойкость при таком подходе значительно ниже.

Впоследствии были построены сети Фейстеля с большим числом потоков. Это позволило также усложнить подход к функциям преобразования. На рисунке 13 представлены примеры таких сетей.

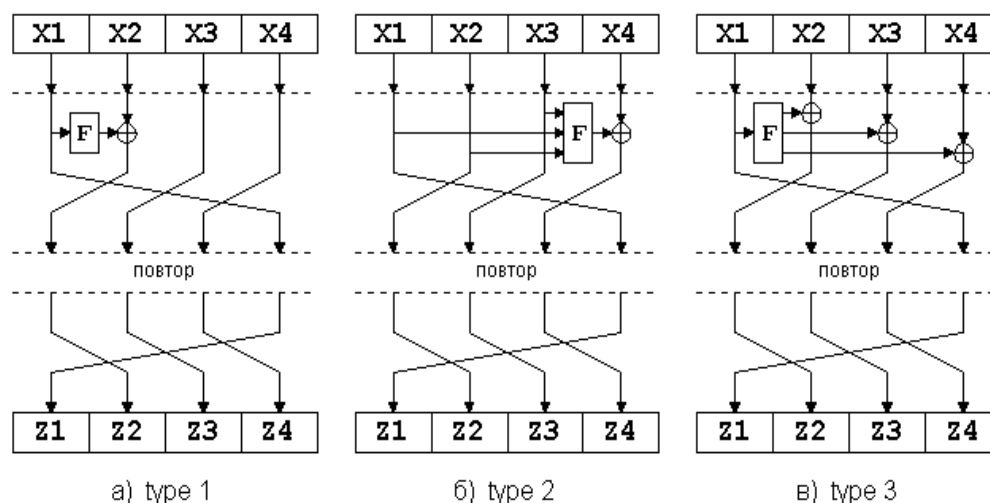


Рисунок 13 - Примеры сетей Фейстеля с большим числом потоков

Также существует несколько способов перемешивания: без него, симметричный, асимметричный, - представленные на рисунке 14.

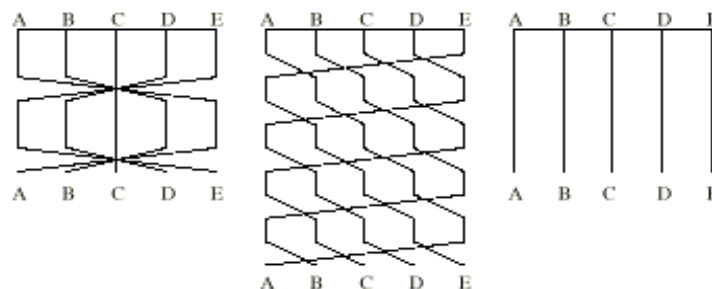


Рисунок 14 - Способы перемешивания потоков

Сравнивая сети Фейстеля с системой Вернама, становится очевидной высокая помехоустойчивость сетей Фейстеля: если система Вернама при искажении одного бита в зашифрованном тексте, искажала один бит в расшифрованном, то в сети Фейстеля этот бит повлияет на довольно большую часть текста и станет заметным, что позволит выявить случайные или преднамеренные искажения, и переслать зашифрованный текст еще раз по другому каналу связи.

В последнее время появились упоминания так называемых Микро Декоррелированных Сетей Фейстеля. Суммируя текст с ключом, криптограф автоматически помещает информацию о ключе в зашифрованный текст, а получение ключа есть конечная цель любого криптоаналитика. Дэвид Вагнер, в своем труде «Атака бумерангом» описал способ уменьшить количество информации в тексте, зашифрованным сетью Фейстеля. Он

предложил рассмотреть функции F_i как функции от текста, так, как если бы ключ был частью функции.

Рассмотрим для примера функцию $f(x)=ax+b$ по одной паре x и $f(x)$ нельзя ничего сказать об a и b . Чем больше пар необходимо, тем лучше такая функция. Но в сетях Фейстеля, криптоаналитик в худшем случае знает исходный текст и зашифрованные тексты, то есть исходные данные и результат последней функции, что делает задачу криптоанализа в случае микро декоррелирующих сетей почти невыполнимой.

Существуют также динамические сети Фейстеля, причем существует несколько их вариантов.

Первый вариант, более простой, предполагает изменение величины блока от раунда к раунду. С одной стороны, это значительно увеличивает время шифрования, так как уже нельзя выбрав блок, сразу полностью произвести все преобразования. С другой стороны, это в значительной мере изменяет статистические характеристики текста. Это легко объяснить тем, что происходит более глубокое перемешивание битов текста. Подстановки в этом случае также несколько отличаются: так как на разных раундах производится подстановка слов различной длины, символы, попадающие в разные блоки, в зависимости от раунда, изменяются по-другому.

Второй вид динамических сетей Фейстеля несколько сложнее. Он использует множества отображающих функций – определенное количество функций, по числу блоков, каждая из которых применяется в своем блоке. При этом множество функций изменяется от раунда к раунду. Это изменение также может быть простым перемешиванием, но в идеале, функции не должны повторяться вообще.

Рассматривают также и объединение двух видов динамических сетей Фейстеля. Когда каждый блок суммируется с результатом своей собственной функции, и количество блоков изменяется в зависимости от раунда.

Задание к лабораторной работе

В соответствии с вариантом задания написать программу, реализующую:

1) Шифрование данных с помощью одного из базовых алгоритмов шифрования (см. варианты заданий). При этом произвести некоторую модификацию выбранного базового алгоритма.

2) Зашифрованные с помощью базового алгоритма данные затем зашифровать с помощью алгоритма «Сеть Фейстеля». В качестве схемы «Сети Фейстеля» взять схему в соответствии с вариантом заданий.

3) В качестве функции F в сети Фейстеля (см. схему сети Фейстеля по варианту заданий) взять XOR с ключом (размер ключа не менее размера блока данных в "Сети Фейстеля").

4) Длина блока для сети Фейстеля - 128 бит, длина потока – 32 бит, кол-во раундов не менее 8.

5) Программа должна выполнять чтение файла, просмотр, его шифрование, расшифрование, просмотр зашифрованного и расшифрованного документа.

Отчет по лабораторной работе должен включать в себя: краткое словесное описание базового алгоритма с его модификацией, схема сети Фейстеля, описание алгоритма работы программы, листинг программы, результаты работы программы.

Варианты заданий:

№ по журналу	Базовый алгоритм	Схема сети Фейстеля
- 1	Шифр Полибия;	рис. 13(а)
- 2	Шифр Полибия;	рис. 13(б)
- 3	Шифр Полибия;	рис. 13(в)
- 4	Шифр Цезаря;	рис. 13(в)

- 5	Шифр Цезаря;	рис. 13(б)
- 6	Шифр Цезаря;	рис. 13(а)
- 7	Шифр Ришелье;	рис. 13(в)
- 8	Шифр Ришелье;	рис. 13(а)
- 9	Шифр Ришелье;	рис. 13(б)
- 10	Перестановочный шифр;	рис. 13(а)
- 11	Перестановочный шифр;	рис. 13(б)
- 12	Перестановочный шифр;	рис. 13(в)
- 13	Шифр Виженера	рис. 13(б)
- 14	Шифр Виженера	рис. 13(а)
- 15	Шифр Виженера	рис. 13(в)
- 16	Шифрование с ключом	рис. 13(в)
- 17	Шифрование с ключом	рис. 13(б)
- 18	Шифрование с ключом	рис. 13(а)
- 19	Шифр Вернама	рис. 13(а)
- 20	Шифр Вернама	рис. 13(б)
- 21	Шифр Вернама	рис. 13(в)

Контрольные вопросы к лабораторной работе:

- 1) Что такое криптоанализ?
- 2) Что такое криптостойкость алгоритма?
- 3) В чем суть перестановочных шифров?
- 4) В чем суть подстановочных шифров?
- 5) Назначение гаммирования.
- 6) Какой из рассмотренных базовых алгоритмов является по Вашему мнению наиболее криптостойким? Объяснить, почему.
- 7) Как, по Вашему мнению, можно увеличить криптостойкость алгоритма с использованием перестановок?

Лабораторная работа №3

Тема: Средства защиты ОС семейства Windows.

Цель: Ознакомиться со стандартными средствами обеспечения защиты ОС семейства Windows на уровне управления пользователями, группами, механизмом квот и использования EFS.

Методические указания к лабораторной работе

1. Создание и управление учетными записями пользователей.

В ОС Windows используются три типа учетных записей пользователей: локальные учетные записи пользователей, учетные записи пользователей домена и встроенные учетные записи пользователей.

- *Локальная учетная запись пользователя (local user account)* позволяет начать сеанс на компьютере и воспользоваться его ресурсами.

- *Учетная запись пользователя домена (domain user account)* позволяет войти в домен и воспользоваться сетевыми ресурсами.

- *Встроенная учетная запись пользователя (built-in user account)* позволяет выполнить административные задачи и воспользоваться локальными или сетевыми ресурсами.

Локальные учетные записи пользователей:

- предоставляют доступ к ресурсам на локальном компьютере;
- создаются исключительно на не входящих в домен компьютерах;
- создаются в базе данных политик безопасности локального компьютера.

Учетные записи пользователей домена

- предоставляют доступ к сетевым ресурсам
- предоставляют маркер доступа для аутентификации
- создаются в службах каталогов Active Directory на контроллере домена

Встроенные учетные записи

Две стандартные встроенные учетные записи:

- администратор;
- гость.

Локальные имена учетных записей пользователей должны быть уникальны в системах, где создаются, а имена учетных записей пользователей домена — уникальны в каталоге домена.

Имена учетных записей могут содержать до 20 символов верхнего или нижнего регистра. В поле разрешается ввести более 20 символов, но ОС Windows признает только первые 20.

Имена учетных записей нечувствительны к регистру, однако ОС Windows сохраняет регистр в целях визуального восприятия.

Пароли могут содержать до 128 символов; минимальная рекомендуемая длина — 8 символов.

При создании паролей используйте символы верхнего и нижнего регистра, цифры и допустимые не алфавитно-цифровые символы.

Создание учетной записи пользователя

Создавать новые учетные записи пользователей могут только администраторы.

Чтобы создать новую учетную запись пользователя, выполните действия, описанные далее.

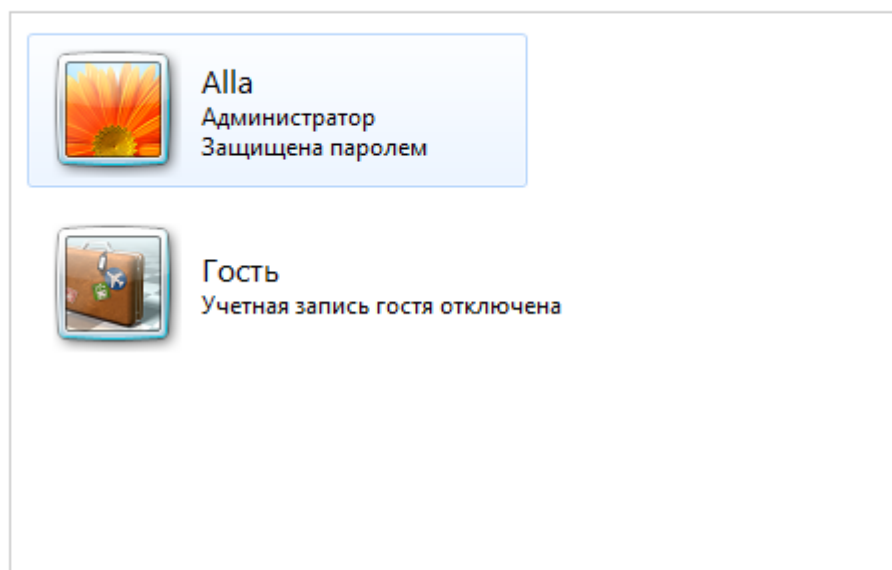
Щелкните Пуск (Start), Панель управления (Control panel) и Учетные записи пользователей (User accounts).

В окне Учетные записи пользователей (User accounts) щелкните Создать новую учетную запись (Create A New Account).

Появляется окно Задайте имя новой учетной записи (Name The New Account).

Выберите нужный тип учетной записи и щелкните Создать учетную запись (Create Account).


Выберите учетную запись для изменения



[Создание учетной записи](#)

[Что такое учетная запись пользователя?](#)

Другие возможные действия

 [Установить родительский контроль](#)

[Переход к начальной странице учетных записей](#)

Рисунок 1 - Изменение свойств существующих учетных записей пользователей

Создание и управление группами пользователей.

Группа (group) — это совокупность учетных записей пользователей. Группы упрощают администрирование, позволяя назначить разрешения и права для группы пользователей, а не индивидуально для каждой учетной записи. Посредством *разрешений* (permissions) можно определить права пользователей при работе с ресурсами типа папки, файла или принтера. Назначив разрешения, вы предоставляете пользователям доступ к ресурсу и определяете тип доступа, который они имеют. Например, если нескольким пользователям необходимо прочесть некий файл, то можно объединить их учетные записи в группу и затем предоставить этой группе разрешение на чтение этого файла. *Права* (rights) позволяют пользователям выполнять

системные задачи, например изменять время, выполнять архивацию и восстановление файлов.

Группа — это совокупность учетных записей пользователей и все члены группы получают разрешения для этой группы.

Пользователи могут быть членами нескольких групп, а группа может входить в другую группу.

Локальная группа (local group) — совокупность учетных записей пользователей на компьютере. Локальные группы используются при назначении разрешений на доступ к ресурсам, постоянно хранящимся на компьютере, где создана локальная группа. ОС Windows создает локальные группы в базе данных локальных политик безопасности. Используйте локальные группы на компьютерах, которые не входят в домен.

Вы можете использовать локальные группы только на компьютере, где они создаются.

Создание локальных групп на примере ОС Windows Server.

Чтобы создать локальные группы в папке **Группы** (Groups), воспользуйтесь оснасткой **Управление компьютером** (Computer Management) (Рисунок 2).

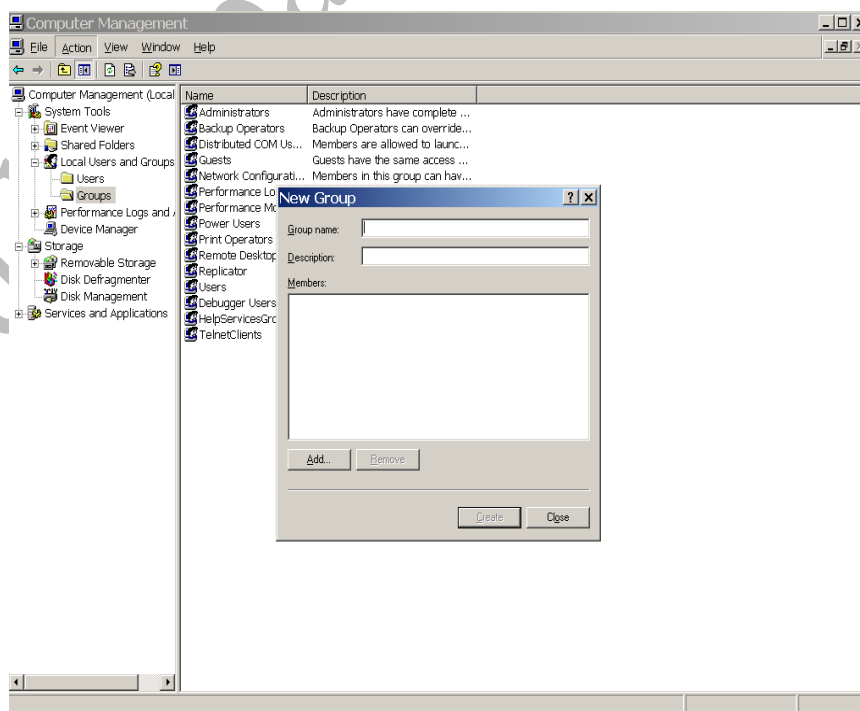


Рисунок 2 – Создание локальных групп пользователей

Чтобы добавить членов в группу с помощью оснастки **Управление компьютером** (Computer Management), выполните действия, описанные далее.

1. Запустите **оснастку Управление компьютером** (Computer Management).

2. Разверните дерево **Локальные пользователи и группы** (Local Users And Groups) и щелкните **Группы** (Groups).

3. В правой области окна щелкните правой кнопкой мыши соответствующую группу, затем щелкните пункт контекстного меню **Свойства** (Properties).

Оснастка **Управление компьютером** (Computer Management) выводит диалоговое окно **Свойства: имя_группы**.

4. Щелкните кнопку **Добавить** {Add}.

Оснастка **Управление компьютером** (Computer Management) выводит диалоговое **окно Выбор: пользователи** (Select Users)

5. Убедитесь, что в текстовом поле **В следующем месте** (From This Location) **выбран** компьютер, где группа создана.

6. В диалоговом окне **Выбор: пользователи** (Select Users), в поле **Введите имена выбираемых объектов** (Enter Object Names To Select), введите **через** точку с запятой имена учетных **записей**, которые хотите добавить в группу, и щелкните **ОК**.

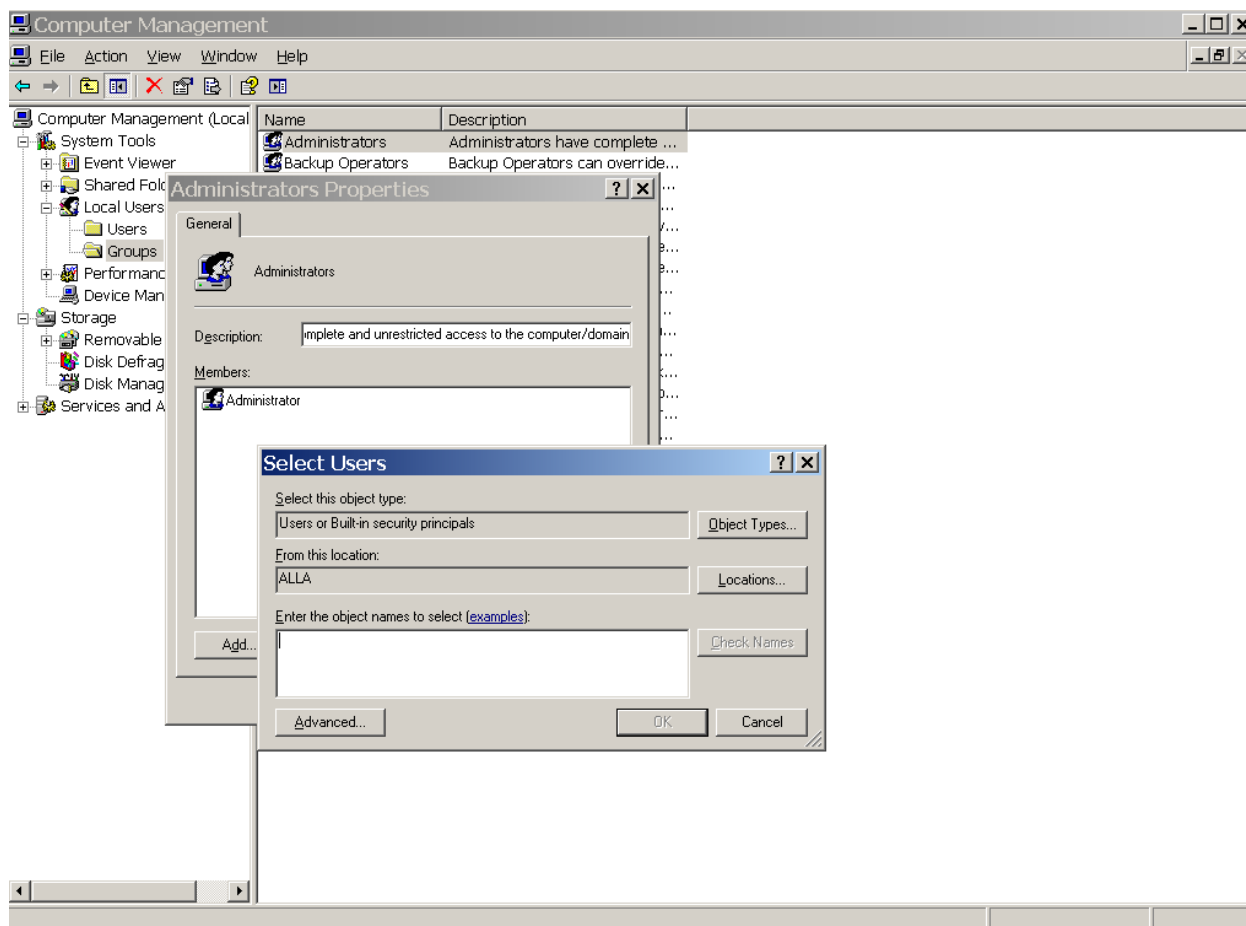


Рисунок 3 – Добавление пользователей в группу на примере ОС Windows Server

2. Обеспечение безопасности с помощью разрешений NTFS.

Разрешения NTFS доступны только на томах NTFS и используются для явного указания того, какие пользователи и группы могут получать доступ к файлам и папкам и какие действия они могут совершать с содержимым этих файлов и папок.

Существуют следующие разрешения NTFS для папок: **Чтение** (Read), **Запись** (Write), **Список содержимого папки** (List Folder Contents), **Чтение и выполнение** (Read & Execute), **Изменить** (Modify) и **Полный доступ** (Full Control).

Существуют следующие разрешения NTFS для файлов: **Чтение** (Read), **Запись** (Write), **Чтение и выполнение** (Read & Execute), **Изменить** (Modify) и **Полный доступ** (Full Control).

Администраторы, владельцы файлов или папок и пользователи, имеющие разрешение **Полный доступ** (Full Control), могут устанавливать разрешения NTFS для других пользователей и групп для управления доступом к файлам и папкам.

В NTFS хранится список управления доступом (ACL), который содержит список всех пользователей и групп, которым был разрешен доступ к файлу или папке, тип установленного доступа для каждого файла и папки на томе NTFS.

Пользователь, пытающийся получить доступ к ресурсу, должен иметь разрешение соответствующего типа.

Вы можете устанавливать множественные разрешения отдельным пользователям или группам при помощи установки разрешений отдельному пользователю и каждой группе, членом которой является пользователь.

Разрешения **NTFS** для файлов имеют больший приоритет, чем разрешения NTFS для папок.

Эффективные разрешения пользователя для ресурса основаны на разрешениях NTFS, назначаемых отдельному пользователю и всем группам, к которым он принадлежит.

3. Управление дисковыми квотами

Дисковые квоты используются для управления объемом хранимых данных в распределенных средах.

Дисковые квоты позволяют распределять дисковое пространство между пользователями в зависимости от того, владельцами каких файлов и папок они являются.

ОС Windows учитывает дисковые квоты для каждого тома, даже если эти тома расположены на одном и том же жестком диске. Поскольку квоты отслеживаются для пользователя, дисковое пространство, занятое файлами пользователя, вычисляется независимо от того, в каких папках пользователь хранит эти файлы.

1) Используемый объем диска вычисляется на основе владения файлами - Когда пользователь копирует или сохраняет файл на томе NTFS либо становится владельцем файла на томе NTFS, Windows XP Professional берет необходимое для размещения файла пространство из выделенной квоты

2) Дисковые квоты не учитывают сжатие - Пользователь отвечает за каждый распакованный байт независимо от того, какой объем дискового пространства используется в действительности. Одна из причин этого заключается в том, что степень сжатия зависит от типа файлов.

3) Свободное пространство для приложений зависит от предела квоты – выделяется остаток дисковой квоты.

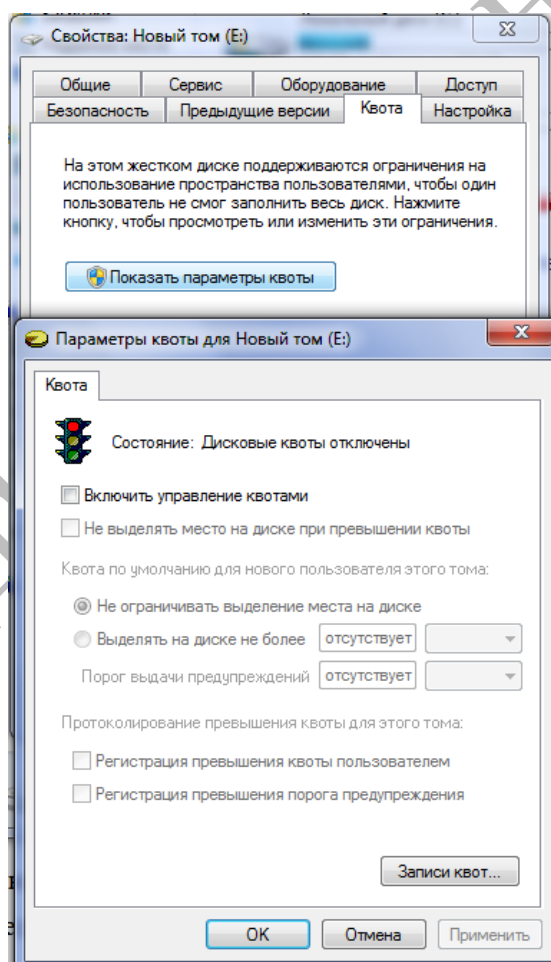


Рисунок 4 – Квоты пользователей

С помощью диалогового окна *Записи квот (Quota Entries For)* можно просмотреть следующую информацию:

- объем дискового пространства, используемый каждым из пользователей;
- значения порога предупреждения и предела квоты для каждого пользователя;
- пользователи, превысившие порог предупреждения;
- пользователи, превысившие предел квоты.

2. Использование EFS.

Файловая система Microsoft Encrypting File System (EFS) предоставляет возможность шифрации данных, хранящихся на дисках NTFS.

Encrypting File System (EFS) –не поддерживается в "домашних" версиях.

EFS использует симметричное шифрование для защиты файлов, а также ассиметричное шифрование для защиты случайно-сгенерированного ключа шифрования для каждого файла.

1) Каждый файл шифруется с помощью симметричного алгоритма шифрования (более быстрый способ шифрования).

Используется случайно-сгенерированный ключ для каждого файла, называемый File Encryption Key (FEK),

2) FEK - (случайный для каждого файла ключ симметричного шифрования) защищается путём ассиметричного шифрования на базе алгоритма RSA.

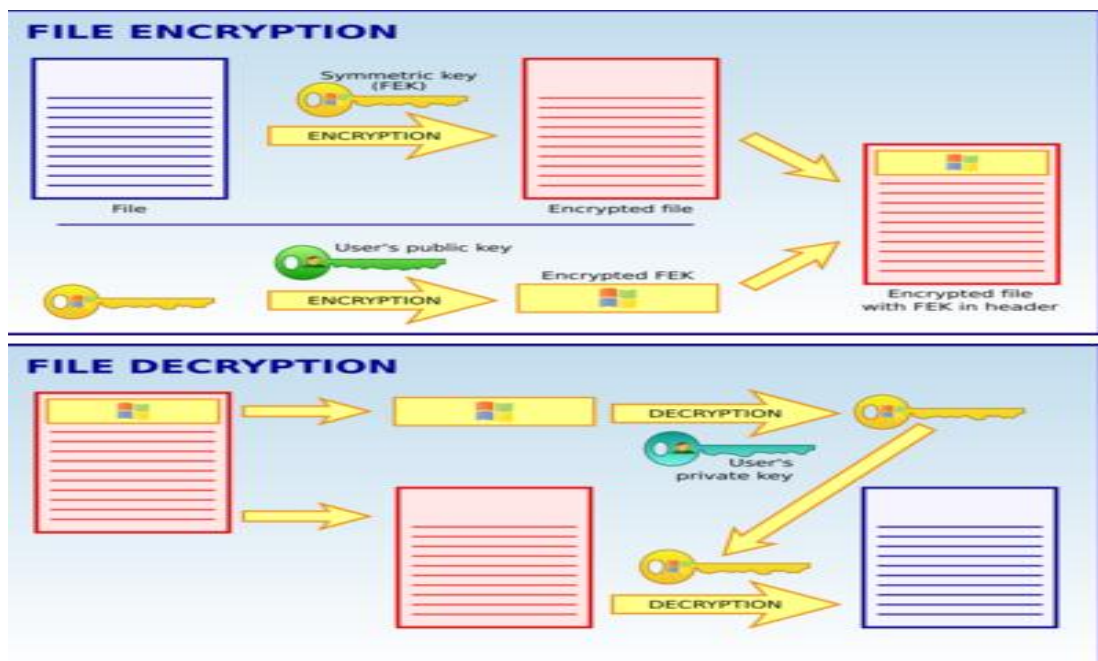


Рисунок 5 - Схема шифрования EFS

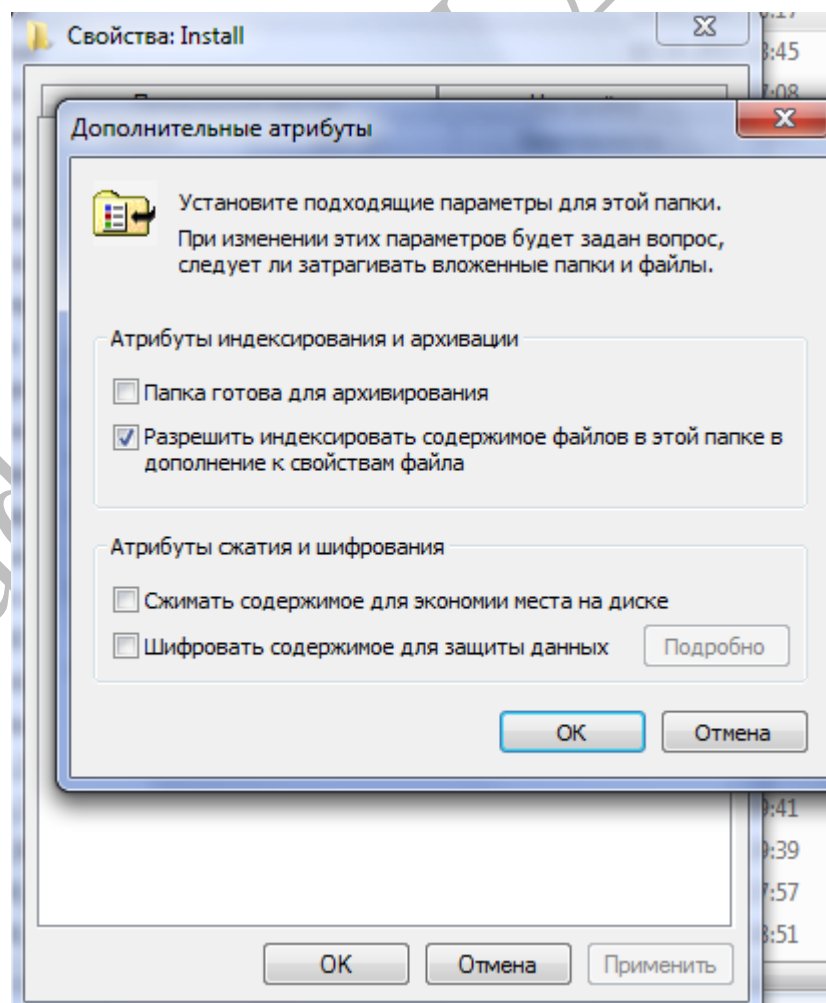


Рисунок 6 - Пример шифрования файла средствами EFS

Задание к лабораторной работе №3

I. Безопасность на уровне файлов и папок.

1) Выполните преобразование тома FAT или FAT32 в файловую систему NTFS без форматирования (если на жестком диске используется NTFS, преобразуйте ФС флэшки, предварительно скопировав данные на жесткий диск перед конвертацией, используя команду CONVERT).

2) Создайте 2 пользователя в системе без назначения им администраторских прав.

3) Создайте 2 группы пользователей без назначения им администраторских привилегий.

4) Каждого пользователя включите в отдельную группу.

5) Приведите примеры на уровне созданных пользователей и групп множественные разрешения NTFS:

- приоритет разрешений для файлов над разрешениями для папок
- приоритет запрещения над разрешениями
- наследование разрешений в NTFS.

II. Дисковые квоты.

1) Для созданных пользователей определите дисковые квоты (объем выделенного пространства на Ваше усмотрение)

2) С помощью диалогового окна Записи квот (Quota Entries For) посмотрите следующую информацию:

- объем дискового пространства, используемый каждым из пользователей;
- значения порога предупреждения и предела квоты для каждого пользователя;
- пользователи, превысившие порог предупреждения;
- пользователи, превысившие предел квоты.

III. Шифрование.

Используя стандартные средства ОС Windows зашифруйте несколько текстовых файлов под администратором, и отобразите результат доступа к этим файлам обычных пользователей, зарегистрированных в системе.

В отчет по лабораторной работе включить экранные формы выполненных действий с кратким описанием выполненных действий.

Контрольные вопросы к лабораторной работе №3

1. Какие типы учетных записей пользователей Вы знаете?
2. Для чего используются группы пользователей.
3. Множественные разрешения NTFS. Назначение. Возможности. Наследование.
4. Назначение дисковых квот. Приведите пример использования дисковых квот.
5. Где хранятся ключи шифрования файлов в EFS? Какие действия необходимо выполнить, если на диске есть зашифрованные файлы EFS и Вам необходимо переустановить операционную систему?
6. В каких целях можно использовать утилиту convert?

Лабораторная работа №4

Тема: Изучение стандартных средств для реализации приложений, использующих симметричное и асимметричное шифрование.

Цель: Изучить модель криптографии .NET Framework, Основные классы и структуры данных, разработать приложение для шифрования файлов использующих симметричные и асимметричные алгоритмы шифрования.

Методические указания у выполнению лабораторной работы

1. Модель криптографии .NET Framework
2. Основные классы и структуры данных
3. Пример приложения для шифрования и дешифровки данных
4. Перечень ссылок

1. Модель криптографии .NET Framework

.NET Framework предоставляет реализацию многих стандартных криптографических алгоритмов. Эти алгоритмы просты в использовании и по умолчанию имеют наиболее безопасные из возможных значений свойств. Кроме того, в .NET Framework имеется криптографическая модель наследования объектов, поточно-ориентированный подход к разработке.

Наследование объектов

Система безопасности .NET Framework реализует расширяемую модель наследования производных классов. Иерархия имеет следующий вид:

- класс типа алгоритма, например `SymmetricAlgorithm` или `HashAlgorithm`. Это абстрактный уровень;
- класс алгоритма, является производным от класса типа алгоритма, например `RC2` или `SHA1`. Это абстрактный уровень;
- реализация класса алгоритма, который является производным от класса алгоритма, например `RC2CryptoServiceProvider` или `SHA1Managed`. Это уровень реализации алгоритма.

Используя данный шаблон производных классов, можно легко добавить новый алгоритм или новую реализацию существующего алгоритма. Например, для создания нового алгоритма шифрования с открытым ключом можно выполнить наследование от класса `AsymmetricAlgorithm`.

Поточно-ориентированный подход

.NET использует поточно-ориентированный подход для реализации алгоритмов симметричного шифрования и хэширования. Основой такого подхода является класс `CryptoStream`, производный от класса `Stream`. Все основанные на потоках криптографические объекты поддерживают единый стандартный интерфейс (`CryptoStream`) для управления своими частями, ответственными за передачу данных. Благодаря тому, что все эти объекты построены на основе стандартного интерфейса, можно сцеплять вместе различные объекты (например, за объектом, реализующим хэширование, поставить объект, реализующий шифрование) и выполнять ряд операций над данными без использования промежуточных хранилищ данных. Поточная модель также позволяет строить объекты на основе меньших объектов. Например, связанные вместе алгоритмы шифрования и хэширования можно рассматривать как единый поточный объект, несмотря на то, что он может быть построен на основе набора некоторых поточных объектов.

2. Основные классы и структуры данных

Класс **`CSParameters`** – содержит параметры, передаваемые поставщику служб шифрования (CSP), который выполняет криптографические вычисления.

Наличие CSP можно установить, проверив с помощью редактора реестра следующий раздел реестра:

HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.

Класс `CspParameters` представляет параметры, которые можно передавать управляемым криптографическим классам, использующим службы шифрования (CSP), с помощью интерфейса Microsoft Cryptography API (CAPI). Классы, имена которых заканчиваются на "CryptoServiceProvider", являются оболочками управляемого кода для соответствующего CSP.

Класс `CspParameters` используется для выполнения следующих задач:

- задание конкретного CSP путем передачи типа поставщика свойству `ProviderType` или `ProviderName`. Можно также задать CSP с помощью перегруженной версии конструктора;
- создание контейнера ключей, в котором можно хранить криптографические ключи. Контейнеры ключей предоставляют самый безопасный способ хранения криптографических ключей и позволяют скрыть их от злоумышленников;
- определение с помощью свойства `KeyNumber` типа создаваемого асимметричного ключа: ключ подписи или ключ обмена.

Класс **`RSACryptoServiceProvider`** - выполняет шифрование и дешифрование данных с помощью реализации асимметричного алгоритма RSA, предоставляемого поставщиком служб шифрования (CSP). Позволяет выполнить экспорт, импорт данных асимметричной пары ключей. Поддерживаются ключи длиной от 384 до 16384 бит с приращениями по 8 бит, если установлен Microsoft Enhanced Cryptographic Provider, и ключи длиной от 384 до 512 бит с приращениями по 8 бит, если установлен Microsoft Base Cryptographic Provider.

<http://msdn.microsoft.com/ru-ru/library/system.security.cryptography.rsacryptoserviceprovider.aspx>

Структура **RSAParameters** - представляет стандартные параметры для алгоритма RSA (значения d, e, n, p, q и т.д.)

<http://msdn.microsoft.com/ru-ru/library/system.security.cryptography.rsaparameters.aspx>

Класс **CryptoStream** – определяет поток, который связывает потоки данных с криптографическими преобразованиями.

<http://msdn.microsoft.com/ru-ru/library/system.security.cryptography.cryptostream.aspx>

Класс **RijndaelManaged** – реализует симметричный алгоритм шифрования Rijndael. Поддерживаются ключи длиной 128, 192 и 256 бит.

<http://msdn.microsoft.com/ru-ru/library/system.security.cryptography.rijndaelmanaged.aspx>

3. Пример приложения для шифрования и дешифровки данных

Предлагаемый пример приложения демонстрирует основные принципы шифрования и расшифровки данных.

В приложении используется класс **RijndaelManaged**, симметричный алгоритм, для шифрования и расшифровки данных из файла с использованием автоматически генерируемых объектов **Key** и **IV**.

Используется класс **RSACryptoServiceProvider**, асимметричный алгоритм, для шифрования и дешифровки ключа, используемого в классе **RijndaelManaged**.

Обязательные компоненты:

Пространство имен **System.IO**, **System.Security.Cryptography**

Создание асимметричного ключа

При выполнении данной задачи создается асимметричный ключ, с помощью которого производится шифрование и дешифровка ключа симметричного алгоритма RijndaelManaged. Этот ключ использовался для шифрования содержимого файла, на форме в элементе управления типа "метка" отображается имя контейнера ключа.

Шифрование файла

Для выполнения этой задачи используются два метода: обработчик события для кнопки Encrypt File (buttonEncryptFile_Click) и метод EncryptFile. Первый метод используется для вывода диалогового окна выбора файла и передачи имени файла второму методу, который выполняет шифрование.

Зашифрованное содержимое, ключ и вектор инициализации сохраняются в объект FileStream, который называется пакетом шифрования.

Метод EncryptFile выполняет следующие действия:

- создает объект RijndaelManaged для шифрования содержимого файла;
- создает объект RSACryptoServiceProvider и выполняется шифрование ключа RijndaelManaged;
- использует объект CryptoStream для чтения и шифрования объекта FileStream исходного файла в виде байтовых блоков в объект назначения FileStream для зашифрованного файла;
- определяет длину зашифрованного ключа и вектора инициализации и создает байтовые массивы соответствующей длины;
- записывает ключ, вектор инициализации и значения их длин в пакет шифрования.

Пакет шифрования имеет следующий формат:

- длина ключа, байты 0-3;
- длина вектора инициализации, байты 4-7;

- зашифрованный ключ;
- вектор инициализации;
- зашифрованный текст;

Значения длины ключа и вектора инициализации могут использоваться для определения начальных точек и длин всех частей пакета шифрования, которые затем могут использоваться при расшифровке файла.

Деширование файла

Для выполнения этой задачи используются два метода: обработчик события для кнопки Decrypt File (buttonEncryptFile_Click) и метод DecryptFile. Первый метод используется для вывода диалогового окна выбора файла и передачи имени файла второму методу, который выполняет расшифровку.

Метод Decrypt выполняет следующие действия.

- создает объект симметричного алгоритма RijndaelManaged для расшифровки содержимого;
- считывает первые восемь байтов объекта FileStream зашифрованного пакета в байтовые массивы для получения значений длин зашифрованного ключа и вектора инициализации;
- извлекает ключ и вектор инициализации из пакета шифрования в байтовые массивы;
- создает объект RSACryptoServiceProvider и дешифрует ключ RijndaelManaged.
- использует объект CryptoStream для чтения и расшифровки зашифрованного текста пакета шифрования FileStream в виде байтовых блоков и загрузки их в объект FileStream для расшифрованного файла. По завершении этой операции дешифровка считается выполненной.

Экспорт открытого ключа

В рамках этой задачи ключ, созданный при нажатии кнопки Create Keys, сохраняется в файл. Экспортируются только открытый ключ.

Данная задача воссоздает ситуацию, в которой Алиса предоставляет Бобу открытый ключ, чтобы он мог зашифровывать для нее файлы. Боб и другие лица, имеющие открытый ключ, не смогут расшифровывать их, поскольку они не имеют полной пары ключей.

Импорт открытого ключа

В рамках данной задачи производится загрузка открытого ключа. Этот ключ был создан при нажатии кнопки Export Public Key.

Данная задача воссоздает ситуацию, в которой Боб загружает открытый ключ Алисы, чтобы зашифровывать для нее файлы.

Получение закрытого ключа

В рамках этой задачи контейнеру ключа присваивается имя, соответствующее имени ключа, созданного при нажатии кнопки Create Keys. Контейнер ключа будет содержать полную пару ключей.

Данная задача воссоздает ситуацию, в которой Алиса использует свой закрытый ключ для расшифровки файлов, зашифрованных Бобом.

Тестирование приложения

Создание ключей, шифрование и расшифровка

Нажмите кнопку Create Keys. На форме отобразится имя ключа и будет указано, является ли он полной парой ключей.

1. Нажмите кнопку Export Public Key. Обратите внимание, что при экспорте параметров открытого ключа текущий ключ в памяти не меняется.
2. Нажмите кнопку Encrypt File и выберите файл.
3. Нажмите кнопку Decrypt File и выберите последний зашифрованный файл.

4. Проверьте содержимое расшифрованного файла.
5. Закройте и перезапустите приложение, чтобы протестировать извлечение сохраненных контейнеров ключей в следующем сценарии.

Шифрование с открытым ключом

1. Нажмите кнопку Import Public Key. В метке отобразится имя ключа и будет указано, является ли он открытым.
2. Нажмите кнопку Encrypt File и выберите файл.
3. Нажмите кнопку Decrypt File и выберите последний зашифрованный файл. Эта операция должна завершиться ошибкой, так как необходим закрытый ключ для дешифровки.

В предыдущем сценарии демонстрируется ситуация, когда для дешифрования файла, передаваемого другому лицу, имеется только открытый ключ.

Расшифровка при помощи закрытого ключа

1. Нажмите кнопку GetPrivateKey. В метке отобразится имя ключа и будет указано, является ли он полной парой ключей.
2. Нажмите кнопку Decrypt File и выберите последний зашифрованный файл. Эта операция должна завершиться успешно, так как имеется полная пара ключей для расшифровки.

Назначение элементов управления на форме:

CreateKeys (Создание асимметричного ключа)

Создается пара ключей, состоящая из асимметричного открытого и закрытого ключа, которой присваивается имя контейнера ключа.

Encrypt File (Шифрование файла)

Вывод диалогового окна для выбора файла, подлежащего шифрованию, и шифрование файла.

DecryptFile (Расшифровка файла)

Вывод диалогового окна для выбора зашифрованного файла, подлежащего расшифровке, и расшифровка файла.

GetPrivateKey (Получение закрытого ключа)

Получение полной пары ключей с помощью имени контейнера ключа.

ExportPublicKey (Экспорт открытого ключа)

Сохранение ключа в XML-файл только с открытыми параметрами.

ImportPublicKey (Импорт открытого ключа)

Загрузка ключа из XML-файла в контейнер ключа.

Структура папок при работе с приложением:

В папку c:\temp\encrypt сохраняются зашифрованные файлы и экспортируется открытый ключ.

В папку c:\temp\decrypt сохраняются расшифрованные файлы.

В папке c:\temp\doc находятся исходные файлы данных.

4. Перечень ссылок

1. Службы криптографии <http://msdn.microsoft.com/ru-ru/library/93bskf9z.aspx>
2. Создание криптографического приложения <http://msdn.microsoft.com/ru-ru/library/bb397867.aspx>

Задание к лабораторной работе №4

- 1) Произвести анализ возможных стандартных средств для реализации криптографических алгоритмов (симметричных и ассиметричных алгоритмов, односторонних хеш-функций).
- 2) Выбрать один из симметричных алгоритмов, изучить его реализацию стандартными средствами, изучить принципы реализации ассиметричного алгоритма RSA.
- 3) Разработать приложение с использованием стандартных средств, выполняющее шифрование выбранных файлов:
 - а) с использованием одного из симметричных алгоритмов шифрования (любого на выбор студента в зависимости от средств разработки).
 - б) реализацию шифрования ключа симметричного алгоритма выполнить с использованием ассиметричного алгоритма RSA.
- 4) Разработать приложение с использованием стандартных средств, выполняющих защиту файлов с помощью электронно-цифровой подписи:
 - а) выбрать одну из односторонних хеш-функций для получения дайжеста сообщения;
 - б) выбрать один из ассиметричных алгоритмов шифрования для шифрования дайжеста сообщения (генерацию ключей ассиметричного алгоритма и их импорт провести независимо от первой части задания по шифрованию файлов).

В отчет по лабораторной работе включить:

- 1) Краткое теоретическое описание выбранных алгоритмов шифрования (симметричного и ассиметричного, односторонней хеш-функции).
- 2) Описание структур, методов, которые использовались в программе.

- 3) Текст программы.
- 4) Результаты работы приложения (экранные формы работы программы)

Контрольные вопросы:

- 1) Какие симметричные алгоритмы шифрования Вы знаете?
- 2) Какие ассиметричные алгоритмы шифрования Вы знаете?
- 3) Для чего используется вектор инициализации, каким образом он передается?
- 4) Влияет ли размер ключа на криптосйкость алгоритма?
- 5) Назначение электронно-цифровой подписи?
- 6) Что такое дайжест сообщения?
- 7) Каким образом можно передавать ключи симметричного алгоритма шифрования, если файл зашифрован и необходимо его передать по сети?

Лабораторная работа №5

Тема: "Защита автоматизированной системы"

Цель: Представить модель угрозы и защиты автоматизированной системы.

Методические указания к лабораторной работе

Характеристика наиболее распространенных угроз безопасности АБС

Проведем краткое описание угроз, с которыми наиболее часто приходится сталкиваться на практике.

Несанкционированный доступ. Это один из наиболее распространенных видов компьютерных нарушений. Он заключается в получении пользователем доступа к объекту, на который у него нет разрешения в соответствии с принятой в данной системе политикой безопасности. Несанкционированный доступ становится возможным из-за непродуманного выбора средств защиты, их некорректной установки и настройки, а также при небрежном отношении к защите своих собственных данных.

Незаконное использование привилегий. Любая защищенная система содержит средства, используемые в чрезвычайных ситуациях, или средства, которые способны функционировать с нарушением существующей политики безопасности. Например, иногда на случай внезапной проверки пользователь должен иметь возможность доступа ко всем наборам системы. Обычно эти средства используются администраторами, операторами, системными программистами и другими пользователями, выполняющими специальные функции. Большинство систем защиты в таких случаях используют наборы привилегий, т.е. для выполнения определенной функции требуется определенная привилегия. Обычно пользователи имеют минимальный набор

привилегий, администраторы - максимальный. Наборы привилегий охраняются системой защиты. Несанкционированный (незаконный) захват привилегий приводит к возможности несанкционированного выполнения определенной функции. Расширенный набор привилегий - мечта любого злоумышленника. Он позволит злоумышленнику совершать желаемые для него действия в обход всех мер контроля. Незаконный захват привилегий возможен при наличии ошибок в системе защиты, но чаще всего при халатности в процессе управления системой защиты, в частности при использовании привилегиями. Строгое соблюдение правил управления системой защиты, соблюдение принципа минимума привилегий позволяет избежать такие нарушения.

Атаки "салями". Атаки такого рода более всего характерны для систем, обрабатывающих денежные счета, а потому наибольшую опасность представляют именно для банков и бухгалтерских систем. Принцип атак "салями" построен на том факте, что при обработке счетов используются целые единицы (гривны, копейки, центы), а при начислении процентов нередко получаются дробные суммы. Например, 6,5% годовых от 102,87 долл. за 31 день составит 0,5495726 долл. Любая система округлит эту сумму до 0,55 долл. Однако, если пользователь имеет доступ к счетам или программам их обработки, он может округлить ее в другую сторону - 0,54 долл., а разницу в 1 цент записать на свой счет. Владелец счета вряд ли заметит ее, а если и обратит внимание, то спишет ее на погрешности обработки и не придаст значения. Злоумышленник же получит прибыль в один цент. При обработке 1000 счетов в день его прибыль составит 100 долл., т.е. примерно 30 000 долл. в год. Но многие банки обрабатывают еще большее количество счетов в день. Отсюда и происходит название таких атак - как колбаса салями изготавливается из небольших частей разных сортов мяса, так и счет злоумышленника пополняется за счет различных других счетов. Если "доход" злоумышленника составляет 50-100 долл. в день, то для

него имеет смысл рисковать. Таким образом, атаки "салями" опасны в основном для крупных банков и других финансовых организаций. Успех таких атак зависит не от величины суммы, так как для любого счета погрешность атаки одинакова, а от количества счетов. Причинами атак "салями" являются, во-первых, погрешности вычислений, позволяющие трактовать правила округления в ту или иную сторону, а во вторых, огромные объемы вычислений. Атаки "салями" довольно трудно распознаются, если только злоумышленник не начинает накапливать на одном счете крупные суммы. Предотвратить такие атаки можно только обеспечением целостности или корректности прикладных программ, обрабатывающих счета, разграничением доступа пользователей АБС к счетам, а также постоянным контролем счетов на предмет утечки сумм.

"Маскарад". Под "маскарадом" понимается выполнение каких-либо действий одним пользователем АБС от имени другого пользователя. При этом такие действия другому пользователю могут быть разрешены. Нарушение заключается в присвоении прав и привилегий. Такие нарушения называются симуляцией или моделированием. Цели "маскарада" - скрытие каких-либо действий за именем другого пользователя или присвоение прав и привилегий другого пользователя для доступа к его наборам данных или для использования его привилегий. Примером "маскарада" может служить вход в систему под именем или паролем другого пользователя, при этом система защиты не сможет распознать нарушение. В этом случае "маскараду" обычно предшествует взлом системы или перехват пароля. Могут быть и другие способы реализации "маскарада", например создание и использование программ, которые в определенном месте могут изменить определенные данные, в результате чего пользователь получает другое имя. "Маскарадом" называют также передачу сообщений в сети от имени другого пользователя. Наиболее опасен "маскарад" в банковских и бухгалтерских системах электронных платежей, где неправильная идентификация клиента может

привести к огромным убыткам. Особенно это касается платежей с использованием электронных карт. Используемый в них метод идентификации с помощью персонального идентификатора достаточно надежен. Но могут происходить нарушения вследствие ошибок его использования. Примитивные примеры этого:

- утеря кредитной карточки;
- использование очевидного идентификатора (своего имени и т.д.)

Для предотвращения "маскарада" необходимо использовать надежные методы идентификации, блокировку попыток взлома системы, контроль входов в нее. Необходимо фиксировать все события, которые могут свидетельствовать о "маскараде", в системном журнале для его последующего анализа.

"Взлом системы". Под "взломом системы" понимают умышленное проникновение в систему, когда взломщик не имеет санкционированных параметров для входа. Способы взлома могут быть различными, и при некоторых из них происходит совпадение с ранее описанными угрозами. Объектом охоты часто становится пароль другого пользователя. Пароль может быть вскрыт, например, путем перебора возможных паролей. "Взлом системы" можно осуществить также, используя ошибки программы входа. Таким образом, основную нагрузку на защиту системы от "взлома" несет программа входа. Алгоритм ввода имени и пароля, их шифрование, правила хранения и смены паролей не должны содержать ошибок. Противостоять "взлому системы" поможет, например, ограничение попыток неправильного ввода пароля (т.е. исключить достаточно большой перебор) с последующей блокировкой терминала и уведомлением администратора в случае нарушения. Кроме того, администратор безопасности должен постоянно контролировать активных пользователей системы: их имена, характер работы, время входа и выхода и т.д. Такие действия помогут своевременно установить факт "взлома" и позволяют предпринять необходимые действия.

"Люк". "Люк" - скрытая недокументированная точка входа в программный модуль. "Люк" вставляется в программу обычно на этапе отладки для облегчения работы: данный модуль можно будет вызвать в разных местах, что позволяет отлаживать отдельные части программы независимо. Наличие "люка" позволяет вызывать программу нестандартным образом, что может серьезно сказаться на состоянии системы защиты. "Люки" могут оказаться в программе по разным причинам. Их могли забыть убрать, они могли быть оставлены для:

- использования при дальнейшей отладке;
- обеспечения поддержки готовой программы;
- реализации тайного доступа к данной программе после ее установки.

Первый из перечисленных случаев - ненамеренный промах, который может привести к бреши в системе защиты. В двух последующих случаях "люки" оставляются намеренно, но они создают серьезные испытания для системы защиты, с которыми она может и не справиться. А вот последний случай может стать первым шагом преднамеренного проникновения к использованию данной программы. Большая опасность "люков" компенсируется высокой сложностью их обнаружения, так как их обнаружение - результат случайного и трудоемкого поиска. Защита от "люков" одна - не допускать появления "люков" в программе, а при приеме программных продуктов - проводить анализ исходных текстов программ с целью обнаружения "люков".

Вредоносные программы. Использование вредоносных программ для воздействия на вычислительные системы приобрело в последние годы большой размах. Под вредоносными программами понимают такие программы, которые прямо или косвенно дезорганизуют процесс обработки информации или же способствуют утечке и искажению информации. Самые

распространенные виды подобных программ называют: "троянский конь", "вирус", "червь", "жадная программа", "захватчик паролей". С существованием таких программ все сейчас знакомы, однако чаще привыкли объединять их под понятием "компьютерный вирус". Приведем деление этих программ на виды.

"Троянский конь" - программа, выполняющая в дополнение к основным, т.е. запроектированным и документированным действиям, действия дополнительные, но не описанные в документации. Аналогия с древнегреческим "троянским конем" оправдана - и в том, и в другом случае в не вызывающей подозрения оболочке таится угроза. "Троянский конь" представляет собой дополнительный блок команд, тем или иным образом вставленный в исходную безвредную программу, которая затем передается пользователям АБС. Этот блок команд может срабатывать при наступлении некоторого условия (даты, времени, команды извне и т.д.). Запустивший такую программу подвергает опасности как свои файлы, так и всю АБС в целом. "Троянский конь" действует обычно в рамках полномочий одного пользователя, но в интересах другого пользователя или вообще постороннего человека, установить личность которого практически невозможно. Наиболее опасные действия "троянский конь" может выполнять, если запустивший его пользователь обладает расширенным набором привилегий. В таком случае злоумышленник, составивший и внедривший "троянского коня", и сам этими привилегиями не обладающий, может выполнять несанкционированные привилегированные функции чужими руками. Для защиты от этой угрозы желательно, чтобы привилегированные и непривилегированные пользователи работали с экземплярами прикладных программ, которые должны храниться и защищаться индивидуально. А радикальным способом от этой угрозы является создание замкнутой среды использования программ.

"Вирус" - программа, которая может заражать другие программы путем включения в них возможно модифицированной копии, которая в свою

очередь сохраняет способность к дальнейшему размножению. Считается, что вирус характеризуется двумя основными особенностями:

1. Способностью к самовоспитанию;
2. Способностью к вмешательству в вычислительный процесс (т.е. к получению возможности управления).

Наличие этих свойств является аналогом "паразитирования" в живой природе, которое свойственно биологическим вирусам. В последние годы проблема борьбы с вирусами стала весьма актуальной, поэтому очень многие занимаются ею. Используются различные организационные меры, новые антивирусные программы и пропаганда всех этих мер. В последнее время удавалось более или менее ограничить масштабы заражений и разрушений. Однако, как и в живой природе, полный успех в этой борьбе не достигнут.

"Червь" - программа, распространяющаяся через сеть и не оставляющая своей копии на магнитном носителе. "Червь" использует механизмы поддержки сети для определения узла, который может быть заражен. Затем с помощью тех же механизмов передают свое тело или его часть на этот узел и либо активизируется, либо ждет для этого подходящих условий. Подходящей средой для распространения "червя" является сеть, все пользователи которой считаются дружественными и доверяют друг другу, а защитные механизмы отсутствуют. Наилучший способ защиты от "червя" - принятие мер предосторожности против несанкционированного доступа к сети.

"Захватчик паролей" - это программы специально предназначенные для воровства паролей. Одна из характерных картин этой процедуры такая: при попытке обращения пользователя к терминалу системы на экран терминала выводится информация, необходимая для окончания сеанса работы. Пытаясь организовать вход, пользователь вводит имя и пароль, которые пересылаются владельцу программы-захватчика, после чего выводится сообщение об ошибке ввода и управление возвращается к операционной системе. Пользователь, думающий, что допустил ошибку при

наборе пароля, повторяет вход и получает доступ к системе. Однако его имя и пароль уже известны владельцу программы-захватчика. Перехват пароля возможен и другими способами. Для предотвращения этой угрозы перед входом в систему необходимо убедиться, что пароль вводится именно системной программе ввода, а не какой-нибудь другой. Кроме того, необходимо неукоснительно придерживаться правил использования паролей и работы с системой. Большинство нарушений происходит не из-за хитроумных атак, а из-за элементарной небрежности. Соблюдение специально разработанных правил использования паролей - необходимое условие надежной защиты.

Классификация угроз безопасности АС

Под угрозой безопасности понимается потенциально возможное воздействие, которое может прямо или косвенно нанести урон пользователям или владельцам АС.

Классификация охватывает только умышленные угрозы, а из признаков классификации выбраны лишь некоторые, поскольку определить все множество угроз и способов их реализации не представляется возможным.

По объему атаки (атакой называют реализацию угрозы) воздействию могут подвергаться следующие компоненты АБС:

1. АС в целом - злоумышленник пытается проникнуть в систему для последующего выполнения каких-либо несанкционированных действий. Для этого обычно используют метод "маскарада", перехват или подделку пароля, взлом;
2. Объекты АС - данные или программы в оперативном запоминающем устройстве (ОЗУ) или на внешних носителях, сами устройства системы, как внешние (дисководы, сетевые устройства, терминалы), так и внутренние (ОЗУ, процессор). Воздействие на объекты системы обычно имеет целью доступ к их содержимому

(нарушение конфиденциальности или целостности хранимой информации) или нарушение их функциональности (например, заполнение всей ОЗУ бессмысленной информацией или загрузка компьютера задачей с неограниченным временем выполнения);

3. Субъекты АС, т.е. процессы или подпроцессы пользователей. Целью таких атак является либо прямое воздействие на ход процесса - его приостановка, изменение привилегий, либо обратное воздействие - использование злоумышленником привилегий и характеристик другого процесса в своих целях;
4. Каналы передачи данных (сами каналы, либо пакеты данных, передаваемых по каналу). Воздействие на пакеты данных может рассматриваться как атака на объекты сети. Воздействие на каналы - специфический род атак, характерный для сети. К этим атакам могут относиться:
 - прослушивание канала и анализ потока сообщений, т.е. нарушение конфиденциальности передаваемой информации;
 - подмена или модификация сообщений в каналах связи и на узлах ретранслятора, т.е. нарушение целостности передаваемой информации;
 - нарушение правил коммутации и адресации, т.е. нарушение доступности сети.

От состояния объекта атаки в момент ее осуществления во многом зависят результаты атаки и работы по ликвидации ее последствий. Объект атаки может находиться в одном из трех состояний:

1. хранения на машинном носителе в пассивном состоянии. При этом воздействие на объект осуществляется с использованием доступа;
2. передачи - по линии связи между узлами сети или внутри узла. Воздействие предполагает либо доступ к фрагментам передаваемой информации (например, перехват пакетов на ретрансляторе сети), либо просто прослушивание с использованием скрытых каналов;

3. обработки - в тех ситуациях, когда объектом атаки является процесс пользователя.

По способу воздействия на объект атаки различают:

1. непосредственное воздействие на объект атаки (в том числе с использованием привилегий). Например, непосредственный доступ к набору данных, программе, службе, каналу связи и т.д., воспользовавшись какой-либо ошибкой защиты. Такие действия обычно легко предотвратить с помощью средств контроля доступа;
2. воздействие на систему разрешений (в том числе захват привилегий). При этом способе несанкционированные действия выполняются относительно прав пользователя на объект атаки, а сам доступ осуществляется потом законным образом. Примером может служить захват привилегий, что позволяет затем беспрепятственно получить доступ к любому набору данных или программе;
3. опосредованное воздействие (через других пользователей). Например, "маскарад", когда пользователь присваивает себе каким-либо образом полномочия другого пользователя, выдавая себя за него. Другой способ - "использование в слепую", когда один пользователь в своих интересах заставляет выполнить другого необходимые действия (которые для системы защиты не выглядят несанкционированными, поскольку их выполняет пользователь, имеющий на это право), причем последний о них может и не подозревать. Для реализации этой угрозы может использоваться вирус (он выполняет необходимые действия и сообщает тому, кто его внедрил о результате).

Два последних способа, особенно "использование в слепую", чрезвычайно опасны. Для предотвращения подобных действий требуется постоянный контроль как со стороны операторов и администраторов, так и со стороны пользователей за своими собственными наборами данных.

Помимо перечисленных для классификации угроз могут быть использованы признаки: по цели реализации угрозы, по принципу воздействия, по характеру воздействия и др. Многообразие классификаций угроз отражает сложность их определения и способов защиты от них.

Защита автоматизированной системы

Исходные данные: Дана система обработки данных бухгалтерского учета. Внутренними пользователями системы являются десять человек. Семи из этих десяти человек в соответствии с их привилегиями доступна только определенная часть данных, причем эти данные не пересекаются между собой. Все данные этой системы доступны только трем пользователям: администратору, главному бухгалтеру и его заместителю.

Для пользователей, имеющих доступ только к части информации срок действия – 1 месяц, для тех, кому доступна вся информация – 2 недели.

Информация, обрабатываемая в системе:

1. Данные об основных средствах.
2. Данные о производственных запасах.
3. Данные о торговых операциях.
4. Данные о себестоимости продукции и ценах ее реализации.
5. Данные о заработной плате.
6. Данные о расчетах с кредиторами.
7. Данные о расчетах с дебиторами.

Автоматизированные банковские и бухгалтерские системы (АБС), открывая новые возможности в организации всей бухгалтерской и банковской деятельности, в повышении качества и надежности, в то же время становятся одной из наиболее уязвимых сторон безопасности современных бухгалтерий и банков, притягивая к себе злоумышленников, как из числа персонала, так и со стороны.

Острота проблемы обеспечения безопасности АБС возрастает в связи с рядом объективных причин. Основной является высокий уровень доверия к

АБС. Им доверяют самую ответственную работу (выполнение финансовых операций), от качества которой зависит функционирование предприятий, а также жизнь и благосостояние многих людей.

Хранимая и обрабатываемая в бухгалтерских и банковских системах информация представляет собой реальные деньги. На основании этой информации производятся расчеты, открываются кредиты, переводятся значительные суммы. Как правило, такая информация конфиденциальна. Вполне вероятно, что незаконное манипулирование хотя бы малой частью этой информации может привести к серьезным убыткам.

Проблема безопасности бухгалтерской и банковской информации усложняется в связи с развитием и распространением сетей ЭВМ. Распределенные системы и системы с удаленным доступом выдвинули на первый план вопрос защиты обрабатываемой и передаваемой информации.

Доступность средств вычислительной техники прежде всего ПЭВМ, привела к распространению компьютерной грамотности в широких слоях населения. Это вызвало многочисленные попытки вмешательства в работу государственных и коммерческих систем, как со злым умыслом, так и чисто из "спортивного" интереса. Многие из этих попыток имели успех и нанесли значительный урон владельцам информации. Поэтому понятна необходимость обезопасить информационные технологии в финансовых и банковских структурах, где хранится ценная информация.

Под безопасностью АБС понимается ее защищенность от случайного или преднамеренного вмешательства в нормальный процесс ее функционирования, а также от попыток хищения, модификации или разрушения ее компонентов. *Защита* - это своего рода соревнование обороны и нападения: кто больше знает, предусматривает действенные меры, тот и выигрывает.

Безопасность АБС достигается обеспечением конфиденциальности обрабатываемой ею информации, а также целостности и доступности компонентов и ресурсов системы.

Конфиденциальность информации - это свойство информации быть известной только допущенным и прошедшим проверку субъектам системы (пользователям, программам, процессам и т.д.). Такие субъекты называются авторизованными. Для остальных объектов системы этой информации как бы не существует.

Целостность компонента (ресурса) - это свойство компонента быть неизменным в семантическом смысле при функционировании системы.

Доступность компонента (ресурса) системы - свойство компонента быть доступным для использования авторизованными субъектами в любое время.

Цель любых мероприятий по обеспечению безопасности АБС является защита владельца и законных пользователей АБС от нанесения им материального или морального ущерба в результате случайных или преднамеренных воздействий на систему.

Различают внешнюю и внутреннюю безопасность АБС.

Внешняя безопасность включает как защиту от случайных внешних воздействий, так и защиту от несанкционированного доступа к информации и любых несанкционированных действий.

Внутренняя безопасность связана с регламентацией деятельности пользователей АБС и обслуживающего персонала, с организацией дисциплины прямого или косвенного доступа к ресурсам системы и к информации.

В ряду всех возникающих проблем центральной является именно защита информации.

Один из подходов к организации АБС - путь создания интегрированных систем обработки данных (ИСОД). Этот путь часто обеспечивает минимальную стоимость создания и функционирования АБС, так как используются коллективные ресурсы для всех ее пользователей, включающие аппаратные и программные средства обработки информации, средства ее хранения и т.д. Удачно выбранные организация и возможности

коллективного ресурса (объем памяти быстроедействие центральной ЭВМ и т.д.) значительно снижают стоимость создания и эксплуатации систем.

Однако использование возможностей коллективного ресурса не означает, что ресурсы должны быть доступны каждому пользователю. Доступ должен быть санкционированным. Доступность определяется теми правилами, которые формулируются при создании АБС. Реализация всех требований санкционированного доступа приводит к некоторому увеличению стоимости создания и реализации систем.

Использование ПЭВМ, включение их в состав локальных вычислительных сетей, а тем более подключение к глобальным сетям усложняют постановку и реализацию обеспечения безопасности АБС. Трудности связаны с обеспечением:

1. Сохранности информации, как в памяти ЭВМ, так и на носителях, хранящихся отдельно от ЭВМ;
2. Достоверности информации при передаче информации по каналам связи;
3. Идентификации принимаемой информации и т.д.

При любом подходе к организации АБС меры по обеспечению безопасности вызывают некоторые неудобства. Главные из них следующие:

1. Дополнительные трудности работы с большинством защищенных систем;
2. Увеличение стоимости защищенной системы;
3. Дополнительная нагрузка на системные ресурсы, что требует увеличения рабочего времени для выполнения одного и того же задания в связи с замедлением доступа к данным и выполнением операций в целом;
4. Необходимость привлечения дополнительного персонала, отвечающего за поддержание работоспособности системы

Анализ риска и составление планов защиты информации

Стоимостное выражение оценки вероятного события, ведущего к потерям, называют риском. Оценки степени риска в случае осуществления того или иного варианта угроз, выполняемых по специальным методикам, называют анализом риска. В процессе анализа риска изучают компоненты АБС, которые могут подвергнуться угрозам, определяют уязвимые места системы, оценивают вероятность для каждой конкретной угрозы и ожидаемые размеры потерь, выбирают возможные методы защиты и просчитывают их стоимость. На заключительном этапе оценивается выгода от применения предполагаемых мер защиты. Эта выгода может иметь как положительный, так и отрицательный знак: в первом случае - очевидный выигрыш, во втором он означает дополнительные расходы на обеспечение собственной безопасности. Исходя из этого анализа, и принимают решения о целесообразности тех или иных мер защиты. В конечном итоге строится план защиты, формируется политика безопасности.

План защиты содержит следующие разделы (группы сведений):

1. Текущее состояние системы.
2. Рекомендации по реализации системы защиты.
3. Ответственность персонала.
4. Порядок ввода в действие средств защиты.
5. Порядок пересмотра плана и состава средств защиты.

Политика безопасности - набор законов, правил и практических рекомендаций, на основе которых строится управление, защита и распределение критичной информации в системе. Она должна охватывать все особенности процесса обработки информации, определяя поведение системы в различных ситуациях. Политика безопасности представляет собой некоторый набор требований, прошедших соответствующую проверку, реализуемых при помощи организационных мер и программно-технических средств, определяющих архитектуру системы защиты. Для конкретных организаций политика безопасности должна быть индивидуальной, зависимой от конкретной технологии обработки информации, используемых

программных и технических средств, расположения организации и т.д. В конечном итоге должна быть произведена оценка надежности системы защиты, т.е. уровня обеспечиваемой ею безопасности. Для оценки степени безопасности в разных странах разработаны критерии оценки безопасности систем, создаются соответствующие стандарты. За разработку этих документов и проверку средств разграничения доступа на соответствие им отвечают различные организации.

Реализация политики безопасности требует настройки средств защиты, управления системой защиты и осуществления контроля функционирования АБС. Как правило, задачи управления и контроля решаются административной группой, состав и размер которой зависят от конкретных условий. Очень часто в эту группу входят администратор безопасности, менеджер безопасности и операторы.

Обеспечение и контроль безопасности представляют собой комбинацию технических и административных мер. По данным зарубежных источников, у сотрудников административной группы обычно $\frac{1}{3}$ времени занимает техническая работа и около $\frac{2}{3}$ - административная (разработка документов, связанных с защитой АБС, процедур проверки системы защиты и т.д.). Разумное сочетание этих мер способствует уменьшению вероятности нарушений политики безопасности.

Административную группу иногда называют группой информационной безопасности. Эта группа может быть организационно слита с подразделением, обеспечивающим внутримашинное информационное обеспечение, с администратором АБС. Но чаще всего она обособлена от всех отделов или групп, занимающихся управлением самой АБС, программированием и другими относящимися к системе задачами во избежание возможного столкновения интересов.

В обязанности входящих в эту группу сотрудников должно быть включено не только исполнение директив вышестоящего руководства, но и участие в выработке решений по всем вопросам, связанным с процессом

обработки информации с точки зрения обеспечения его защиты. Все их распоряжения, касающиеся этой области, обязательны к исполнению сотрудниками всех уровней и организационных звеньев банка.

Задание к лабораторной работе

1. Представить план угрозы несанкционированного доступа к системе и описать средства защиты от этой угрозы.
2. Разработать модель угрозы незаконного использования привилегий и описать средства защиты от этой угрозы.
3. Представить модель атаки "салями" и описать средства защиты от этой атаки.
4. Представить модель атаки "маскарад" и описать средства защиты от этой атаки.
5. Представить модель атаки "взлом системы" и описать средства защиты от этой атаки.
6. Представить модель угрозы, используя "люки" и описать средства защиты от этой угрозы.
7. Перечислить возможные дополнительные функции, чтобы данная система стала так называемым "троянским конем". Привести алгоритм работы «троянского коня», описать средства защиты от этой угрозы.
8. Привести модель "захватчика паролей", и описать средства защиты от этой угрозы.
9. Привести алгоритм несанкционированного входа в систему, защита которой основана на фрагментарном подходе, и описать средства защиты от этой угрозы.
10. Привести алгоритм несанкционированного входа в систему, защита которой основана на комплексном подходе, и описать средства защиты от этой угрозы.

11. Привести алгоритм проникновения в систему, когда объектом атаки является система в целом, и описать средства защиты от этой угрозы.
12. Привести алгоритм проникновения в систему, когда объектом атаки является объект системы, и описать средства защиты от этой угрозы
13. Привести алгоритм проникновения в систему через непосредственное воздействие на объект атаки, и описать средства защиты от этой угрозы
14. Привести алгоритм проникновения в систему через воздействие на систему разрешений и описать средства защиты от этой угрозы
15. Представить алгоритм проникновения в систему к данным, которые находятся в состоянии хранения на машинном носителе, и описать средства защиты от этой атаки.
16. Представить алгоритм проникновения в систему, когда целью реализации угрозы является получение имеющейся в системе информации, и описать средства защиты от этой угрозы.
17. Представить алгоритм проникновения в систему, когда целью реализации угрозы является изменение имеющейся в системе информации, и описать средства защиты от этой атаки.

Контрольные вопросы к лабораторной работе

1. Классификация угроз безопасности относительно АБС.
2. Перечислите типы атак, в том числе применимые для АБС.
3. Классификация вирусов и вредоносных программ. Какие типы вирусов можно использовать с целью деструктивных действий в АБС.
4. Что такое защита, основанная на фрагментарном подходе?
5. Что такое защита, основанная на комплексном подходе?
6. Какие средства защиты от сетевых атак Вы знаете?

7. Какие средства защиты от атак, объектом которых является информация, хранящаяся на компьютере, Вы знаете?
8. Что такое политика безопасности?
9. В чем суть аудита предприятия с точки зрения защиты информации?
10. Какие основные пункты должен включать в себя план обеспечения защиты информации на предприятии?

Кафедра ПИ ДонНТУ

Лабораторная работа №6

Тема: Использование стеганографических и криптографических средств защиты информации.

Цель: исследование стеганографических алгоритмов защиты информации, разработка программного продукта встраивания информации в изображение с использованием стеганографических и криптографических алгоритмов.

Методические указания к выполнению лабораторной работы

В настоящее время информация представляет огромную ценность. С появлением глобальных компьютерных сетей, в частности сети Интернет, доступ к информации значительно упростился, что привело к повышению угрозы нарушения безопасности данных при отсутствии мер их защиты, а именно угрозу несанкционированного доступа к информации при ее хранении и передаче.

Криптография – это наука об обеспечении безопасности данных. Она занимается поисками решений четырех важных проблем безопасности – конфиденциальности, аутентификации, целостности и контроля участников взаимодействия. Шифрование – это преобразование данных в нечитабельную форму, используя ключи шифрования-расшифровки. Шифрование позволяет обеспечить конфиденциальность, сохраняя информацию в тайне от того, кому она не предназначена. Целью криптографии является скрывание содержимого сообщений за счет их шифрования.

Стеганография – это метод организации связи (передачи сообщений), при котором скрывается само наличие связи. Методы стеганографии позволяют встраивать секретные сообщения в открытые послания таким образом, чтобы было невозможным заподозрить существование самого встроенного послания. Целью стеганографии является скрывание самого факта существования секретных данных при их передаче, хранении или обработке.

В отличие от криптографии, которая скрывает содержимое секретного сообщения, стеганография скрывает само его существование. Стеганографию обычно используют совместно с методами криптографии, таким образом, дополняя её.

Основными стеганографическими понятиями являются сообщение и контейнер. Контейнером («stegano») называют несекретные данные, которые используют для сокрытия сообщений. В качестве контейнеров могут быть использованы различные оцифрованные данные: растровые графические изображения, цифровой звук, цифровое видео, всевозможные носители цифровой информации, а также текстовые и другие электронные документы. Сообщением называют секретную информацию, наличие которой в контейнере необходимо скрыть. Сообщение также представляет собой некоторый электронный документ. Ключом называют секретную информацию, известную только законному пользователю.

Пустой контейнер (немодифицированный или исходный контейнер) – это некоторый контейнер, не содержащий сообщения. Заполненный контейнер (или соответственно модифицированный контейнер) – это контейнер, содержащий шифрованное сообщение.

Стеганографическим алгоритмом принято называть два преобразования: прямое стеганографическое преобразование и обратное стеганографическое преобразование.

Под стеганографической системой необходимо понимать систему, представляющую собой совокупность сообщений, секретных ключей, контейнеров и связывающих их преобразований.

Под внедрением (сокрытием) сообщения с помощью системы в контейнер понимают применение прямого стеганографического преобразования.

Извлечение сообщения есть не что иное, как применение обратного стеганографического преобразования.

Как правило, выделяют 3 основные цели использования стеганографии:

- цифровые отпечатки (ЦО) (DigitalFingerprint);
- стеганографические водяные знаки (СВЗ) (StegoWatermarking);
- скрытая передача данных (СПД).

Использование стеганографии для цифровых отпечатков предназначено для того, чтобы можно было обнаружить скопирован контейнер или нет. Например, в каждую продаваемую, оригинальную цифровую копию фильма можно вкраплять скрытое сообщение. Если пользователь, купивший экземпляр фильма, опубликует его на каком-нибудь интернет-ресурсе, то с помощью ЦО можно будет идентифицировать злоумышленника. Таким образом, ЦО могут быть применимы для защиты исключительного права.

Задача водяного знака – подтвердить авторскую подлинность защищаемого контейнера. Например, при съемках на видеокамеру можно в каждый кадр вкраплять информацию о времени записи, модели видеокамеры или имени оператора видеокамеры. В случае, если отснятый материал попадет в руки конкурирующей компании, можно попытаться использовать СВЗ для подтверждения авторства записи на суде. Таким образом, СВЗ может быть применим для защиты права авторства.

Скрытая передача данных (СПД) в основном используется в военных и криминальных целях для незаметной передачи данных из одной точки в другую. Например, менеджер одной крупной компании может получить копию базы данных коммерческой информации и внедрить его в аудио или видеофайл.

Каждая из указанных выше задач требует определенного соотношения между устойчивости встроенного сообщения к внешним влияниям и размером встроенного сообщения.

Для большинства современных методов, которые используются для скрытия сообщений файлов цифрового формата, имеет место зависимость надежности системы от объема встраиваемых данных, представленная на рисунке 1.1.

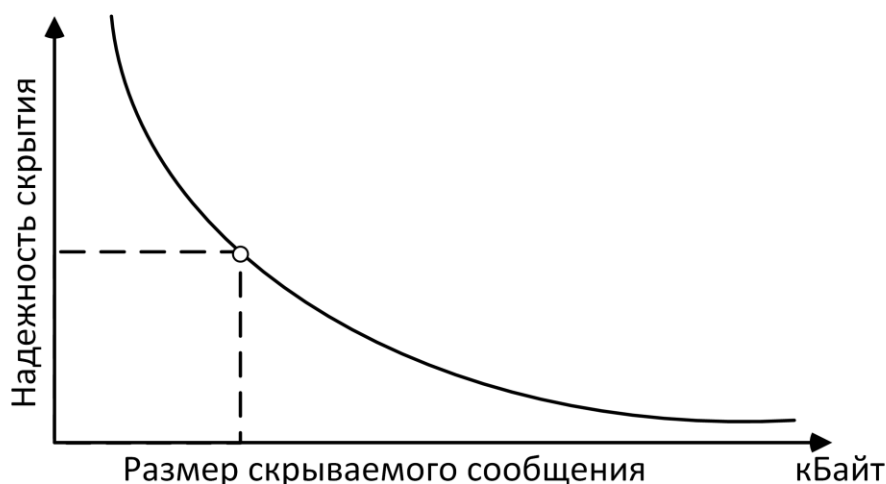


Рисунок 1.1 –

Взаимосвязь между устойчивостью стеганосистемы и объемом скрываемого сообщения при неизменном размере файла-контейнера

Из рисунка 1.1 видно, что увеличение объема встраиваемых данных значительно снижает надежность системы.

Таким образом, существует перспектива принятия оптимального решения при выборе между количеством (объемом) скрываемых данных и степенью устойчивости (скрытости) к возможной модификации (анализу) сигнала-контейнера.

При использовании любого метода, благодаря избыточности информации, существует возможность повысить степень надежности скрытия, жертвуя при этом пропускной способностью (объемом скрываемых данных). Также очевиден и тот факт, что в зависимости от целей, для которых используется скрытие данных, различными являются и требования относительно уровня устойчивости системы к модификации контейнера. Как следствие, для разных целей оптимальными являются разные методы стеганографии.

Характеристика популярных существующих программных средств.

На сегодняшний день существует огромное количество открыто распространяемых программных продуктов, использующих

стеганографическую технику. Наиболее популярны инструменты размещаются в Github-репозиториях.

В рамках проекта DarkJPEG разработан стеганографический веб-сервис нового поколения, позволяющий скрывать конфиденциальную информацию в виде незаметного шума в JPEG-изображениях, при этом выделить данную информацию можно только зная заданный при кодировании секретный ключ-пароль. Сервис использует стойкие методы стеганографии для сокрытия самого факта сокрытия информации вместе со стойкими методами криптографии для защиты данных, передаваемых по открытым каналам.

Основные особенности:

- использование SHA-3 для генерации ключей;
- симметричное шифрование AES-256;
- JPEG (DCT LSB) стеганография;
- поддержка RarJPEG и двойного сокрытия;
- подбор случайного контейнера;
- вычисления без участия сервера (на клиентской стороне);
- гарантия полной конфиденциальности.

OpenStego – это стеганографическое приложение, которое предоставляет две функциональности:

- сокрытие данных (можно скрыть любые данные внутри файла-контейнера);
- водяные знаки (встраивание невидимой подписи, например в изображении, с целью обнаружения несанкционированного копирования).

Данная программа предоставляет возможность надежного встраивания водяных знаков, при незначительном изменении размера или кадрирования изображения. Отличительной особенностью OpenStego является то, что можно расширить ее функционал с помощью своих плагинов, написанных на Java.

LSB-Steganography – стеганографическая программа для сокрытия данных в изображениях, реализована в виде скрипта на языке Python,

использующая методы наименьшего значащего бита. Модуль LSBSteg основан на OpenCV (библиотека для работы с изображениями). Он использует незначащий бит каждого пикселя и каждого цвета изображения для скрытия данных. Программа не выполняет никаких дополнительных криптографических преобразований для защиты внедренных данных.

При проектировании стеганографической системы можно выделить три основных этапа:

- получение из цифрового документа байтов;
- преобразование байтов цифрового документа (подготовка данных для внедрения в контейнер);
- внедрение данных в изображение, используя метод LSB.

На первом этапе некоторый цифровой объект должен быть представлен в виде потока байт. Если это файл, то его можно открыть в бинарном режиме. Если текстовое сообщение, то представить его в виде байтов, с заданной кодировкой.

На втором этапе, на вход подаются байты, полученные на первом этапе. Прежде всего необходимо вычислить контрольную хеш-сумму исходных байт. После этого сжать исходные байты, выбранным пользователем алгоритмом сжатия. К сжатым байтам в начало добавляется заголовок (метаданные). После чего эти байты шифруются симметричным алгоритмом шифрования.

Последний этап состоит в том, что зашифрованные байты внедряются в изображение.

На рисунке 1 схематически показаны три этапа работы стеганографической системы.



Рисунок 1 – Этапы работы программной системы

Обоснованием выбора данной структуры системы может служить то, что любой компонент из каждого этапа можно расширить и/или изменить, не затрагивая другие компоненты системы.

Архитектура самого приложения состоит из пользовательского графического интерфейса, который управляет логикой самого приложения и стеганографической системы. На рисунке 2 показана архитектура приложения.

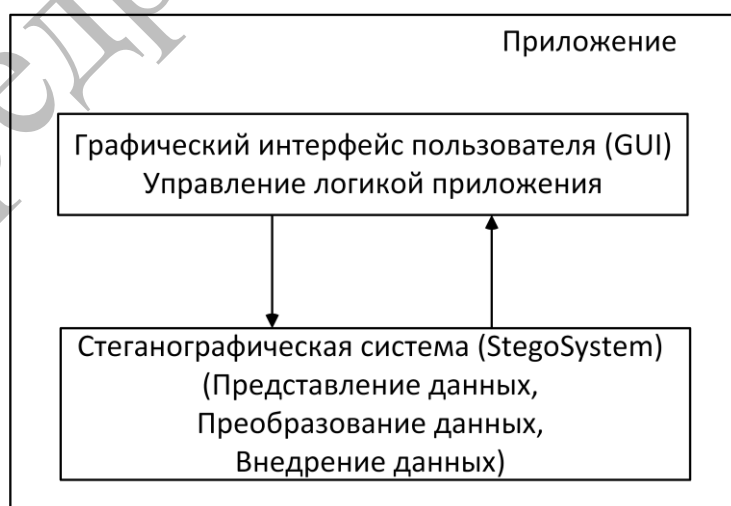


Рисунок 2 – Архитектура приложения

При проектировании системы прежде всего необходимо определить основные требования к программе.

На рисунке 3 показана диаграмма вариантов использования разрабатываемой системы.

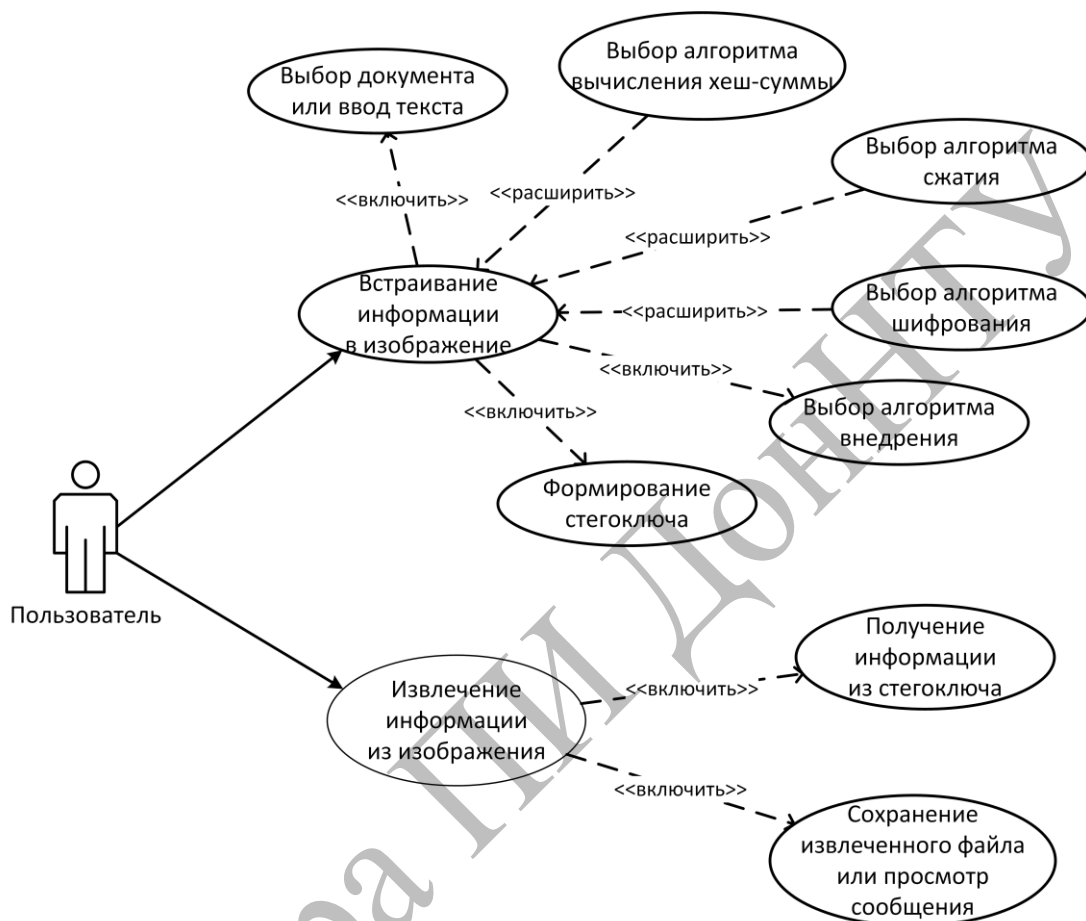


Рисунок 3 – Диаграмма вариантов использования (Uses Case) разрабатываемой системы

Описание структуры формата BMP

BMP– популярный формат несжатого растрового изображения, разработанный компанией Microsoft. Формат BMP является одним из встроенных форматов в ОС Windows. Он поддерживает изображения с 1, 4, 8, 16, 24 и 32 битами на пиксель, хотя файлы BMP с 16 и 32 битами на пиксель используются редко. Для изображений с 4 и 8 битами на пиксель формат BMP поддерживает также простое RLE-сжатие (Кодирование длин серий). Однако сжатие в BMP-формате оказывает эффект только при наличии в

изображении больших областей одинакового цвета, что ограничивает ценность встроенного алгоритма сжатия. BMP-файлы редко находятся в сжатом виде.

В формате BMP многобайтовые целые числа хранятся с младшим байтом на первом месте. Данные, сохраняемые в формате BMP, целиком состоят из полных байтов, поэтому вопрос о порядке битовых строк не возникает.

В таблице 1 показана структура формата BMP-файла.

Таблица 1 – Структура формата BMP

Название поля	Размер
Заголовок файла растровой графики (14 байт)	
Сигнатура файла BMP	2 байта
Размер файла	4 байта
Не используется	2 байта
Не используется	2 байта
Местонахождение данных растрового массива	4 байта
Информационный заголовок растрового массива (40 байт)	
Длина этого заголовка	4 байта
Ширина изображения	4 байта
Высота изображения	4 байта
Число цветовых плоскостей	2 байта
Бит/пиксел	2 байта
Метод сжатия	4 байта
Длина растрового массива	4 байта
Горизонтальное разрешение	4 байта
Вертикальное разрешение	4 байта
Число цветов изображения	4 байта

Число основных цветов	4 байта
Таблица цветов (длина изменяется от 8 до 1024 байт)	
Собственно данные растрового массива (длина переменная)	

На приведенной схеме показана структура типичного BMP-файла, содержащего 256-цветное изображение (с глубиной 8 бит/пиксел). Файл разбит на четыре основные раздела: заголовок файла растровой графики, информационный заголовок растрового массива, таблица цветов и собственно данные растрового массива. Заголовок файла растровой графики содержит информацию о файле, в том числе адрес, с которого начинается область данных растрового массива. В информационном заголовке растрового массива содержатся сведения об изображении, хранящемся в файле, например, его высоте и ширине в пикселях. В таблице цветов представлены значения основных цветов RGB (красный, зеленый, синий) для используемых в изображении цветов.

Описание LSB-алгоритма

Цифровые изображения представляют собой матрицу пикселей. Пиксель – это единичный элемент изображения. Он имеет фиксированную разрядность двоичного представления. Например, пиксели полутонового изображения кодируются 8 битами (значения яркости изменяются от 0 до 255).

Младший значащий бит (LSB) изображения несет в себе меньше всего информации. Известно, что человек обычно не способен заметить изменение в этом бите. Фактически, он является шумом. Поэтому его можно использовать для встраивания информации. Таким образом, для полутонового изображения объем встраиваемых данных может составлять 1/8 объема контейнера. Например, в изображение размером 512x512 можно встроить 32 килобайта информации. Если модифицировать два младших бита

(что также почти незаметно), то можно скрытно передать вдвое больший объем данных.

Достоинства рассматриваемого метода заключаются в его простоте и сравнительно большом объеме встраиваемых данных. Однако, он имеет серьезные недостатки. Во-первых, скрытое сообщение легко разрушить. Во-вторых, не обеспечена секретность встраивания информации. Нарушителю точно известно местоположение всего секретного сообщения.

Задание к лабораторной работе

Изучить существующие программные продукты для стеганографической защиты информации.

Написать программу защиты информации с использованием криптографических (ранее изученных) и стеганографических (LSB) алгоритмов. В качестве контейнера использовать графический формат BMP.

В алгоритме LSB число используемых младших бит четные варианты - 2 бита, нечетные варианты - 1 бит.

В качестве симметричного алгоритма шифрования использовать алгоритм AES. Четные варианты используют режим CBC. Нечетные варианты используют режим CFB.

Для проверки целостности данных использовать однонаправленные хеш-функции. Четные варианты - MD5, нечетные варианты - SHA2 В конец файла хеш добавляют четные варианты, в начало файла - нечетные варианты.

В рамках выполнения лабораторной работы необходимо:

- 1) Считать файл для дальнейшего шифрования симметричным алгоритмом (AES)
- 2) Получить хеш с использованием однонаправленной хеш-функции.
- 3) Добавить хеш к файлу.
- 4) Зашифровать файл симметричным алгоритмом.

- 5) Поместить зашифрованный файл в BMP-контейнер по алгоритму LSB.
- 6) Извлечь данные из BMP-контейнера.
- 7) расшифровать с использованием ключа симметричного алгоритма.
- 8) Проверить целостность данных.

Контрольные вопросы:

- 1) Что такое стеганография?
- 2) Можно ли использовать другие графические форматы в качестве стеганографических контейнеров?
- 3) Каким образом защитить ключ симметричного алгоритма шифрования?
- 4) Как предусмотреть ситуацию, чтобы сообщение полностью уместилось в стеганографический контейнер?
- 5) Какие стеганографические алгоритмы Вы знаете еще?
- 6) Как повысить сложность алгоритма LSB?

Лабораторная работа №7

Тема: «Обфускация программ»

Цель работы: Изучение возможностей существующих обфускаторов для защиты программных продуктов.

Методические указания к лабораторной работе

На сегодняшний день проблема сохранения прав на владение интеллектуальной собственностью стоит очень остро. В то время как настольные приложения могут обеспечить себя различными средствами лицензирования и защиты, веб-приложения не могут защитить свой код, расположенный на стороне клиента от просмотра и изучения.

Частично эту проблему решает перенос особенно важных частей кода на сторону сервера, что повлечёт за собой увеличение нагрузки на сервер и может потребовать увеличения мощности сервера или покупки дополнительного оборудования.

Также решению этой проблемы может поспособствовать обфускация программного кода, передающегося клиенту. Обфускация не решит эту проблему полностью, так как возможность использовать реверс-инжиниринг остаётся, но вот потенциальная сложность деобфускации и большие человеческие ресурсы должны отбить желание у злоумышленников разбирать чужой код и извлекать из него алгоритмы.

Обфускация (от лат. *obfuscare* — затенять, затемнять; и англ. *obfuscate* — делать неочевидным, запутанным, сбивать с толку) или запутывание кода — приведение исходного текста или исполняемого кода программы к виду, сохраняющему его функциональность, но затрудняющему анализ и понимание алгоритмов работы.

Для создания запутанного ассемблерного текста могут использоваться специализированные компиляторы, использующие неочевидные или недокументированные возможности среды исполнения программы.

Существуют также специальные программы, производящие обфускацию, так называемые обфускаторы.

Цели обфускации:

- Демонстрация неочевидных возможностей языка и квалификации программиста (если производится вручную, а не инструментальными средствами).
- Оптимизация программы с целью уменьшения размера работающего кода и (если используется некомпilierуемый язык) ускорения работы.
- Затруднение декомпиляции/отладки и изучения вредоносных программ с целью предотвращения обнаружения вредоносной функциональности.

Для получения более подробных теоретических сведений рекомендуется обратиться к электронному ресурсу:
<http://citforum.ru/security/articles/obfus/>.

Рассмотрим использование обфускации для защиты программного кода скриптов на языке JavaScript

Сохранение исходного кода коммерческих программных продуктов, написанных на JavaScript, является актуальной проблемой на данное время. Исходный код таких продуктов, так или иначе, может быть просмотрен злоумышленниками с помощью обычных программ-браузеров. Есть несколько способов предотвращения утечки информации:

- а) перенос исходного кода на сторону сервера;
- б) проведение минификации кода;
- в) проведение обфускации кода.

Первый способ действует безотказно, но при этом увеличивается нагрузка на сервер. Минификация произведёт достаточно незначительные преобразования и код можно будет разобрать. Обфускация же может преобразовать исходный код так, что уже будет довольно затруднительно разобрать алгоритмы, использованные в коде, и логику программы.

Работа программной системы достаточно проста. Она должна получать на вход исходный код JavaScript, который можно либо набрать в поле ввода, либо загрузить из файла. Далее необходимо выбрать нужные режимы обфускации и запустить процесс запутывания.

После обфускации исходного кода программная система предоставляет возможность извлечь его, сохранив его в выходной файл или просто скопировав из поля вывода.

Обзор существующих обфускаторов для JavaScript

На данный момент существует множество обфускаторов для JavaScript, но необходимо отметить, что некоторые обфускаторы являются в первую очередь упаковщиками или минификаторами (программы, которые минимизируют размер исходного кода, выполняют оптимизацию для увеличения производительности), а уже как следствие, они являются обфускаторами. Вот список самых популярных обфускаторов:

- а) YUI Compressor;
- б) Packer;
- в) Google Closure Compiler;
- г) JavaScript Obfuscator (Canada);
- д) Uglify JS;
- е) ESMangle.

YUICompressor и Packer (см. рис. 1) не являются обфускаторами, которые выполняют сложные преобразования над исходным кодом программы, как упоминалось выше, а только минимизируют размер программного продукта и проводят различные оптимизации. Эти минификаторы имеют только web-интерфейс и можно воспользоваться ими только на их сайтах.

GoogleClosureCompiler представляется отдельным приложением, способным проводить как незначительную обфускацию, так и использовать

продвинутые преобразования. Также этот обфускатор способен отслеживать зависимость между файлами в папке с исходным кодом и обфусцировать совместно всю папку с файлами исходного кода. Однако минусом этого обфускатора является то, что для эффективной и корректной работы программы необходимо писать исходный код в определённом стиле и следовать определённым правилам. Это будет удобно не для всех пользователей, как например, обфускируя уже готового проекта, для которого нет возможности переписать исходный код, особенно если кода достаточно много (несколько десятков тысяч строк).

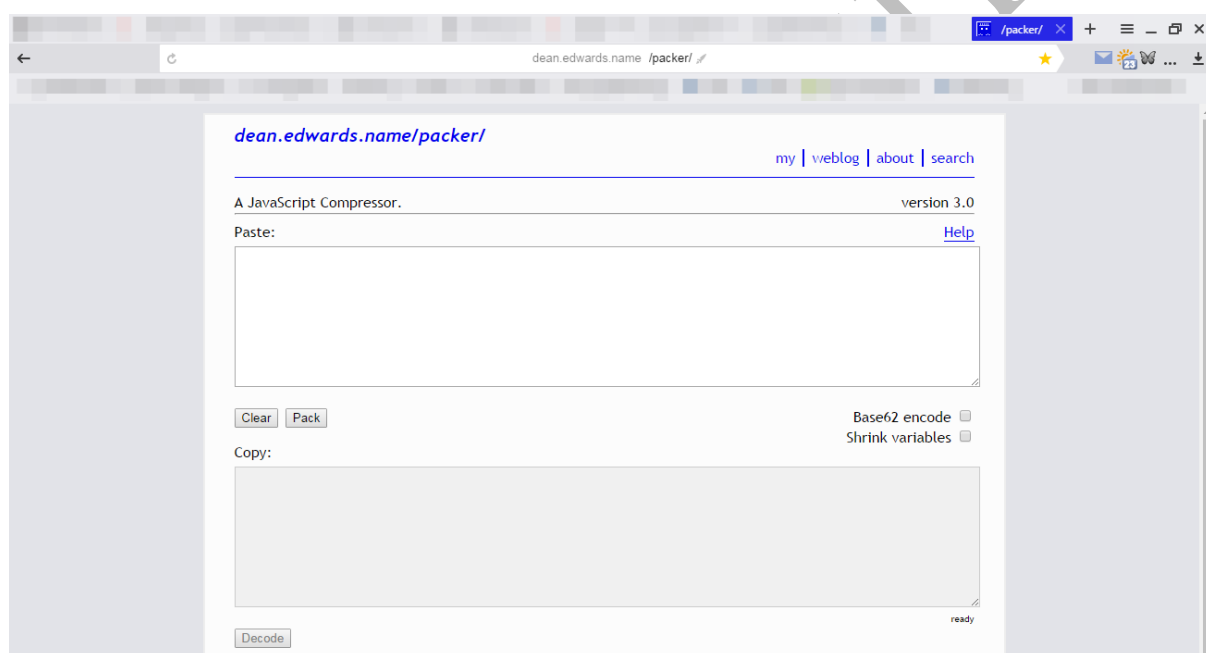


Рисунок 1 – Интерфейс минификатораPacker

Коммерческий продукт JavaScriptObfuscatorv 2.18, который предлагает бесплатную демонстрационную версию, предоставляющую минимальную обфускацию на уровне упаковщиков. Однако в полной версии этот обфускатор предлагает интересные продвинутые способы преобразования кода: кодирование строковых констант, сжатие кода, упрощение глобальных имён и переменных и т.д. На рисунке 2 представлен внешний вид интерфейса программы.

Обфускатор UglifyJS поставляется в виде JavaScript-библиотеки и проводит только минимальные преобразования (удаление форматирования и минимизирование размера исходного кода). Этот обфускатор доступен для установки в виде пакета для Node.js, и его исходный код открыт, и любой может сделать вклад в его разработку. Однако всё-таки, сложных преобразований он не совершает.

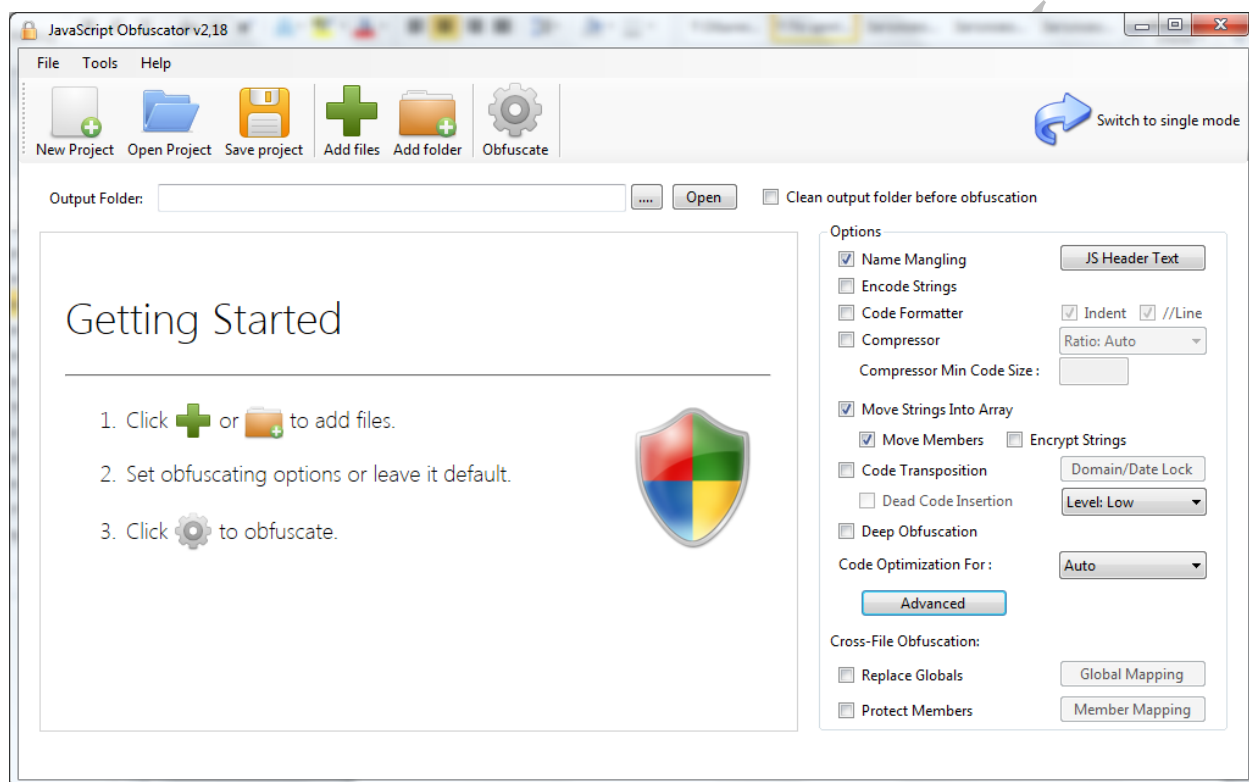


Рисунок 2 – Интерфейс JavaScript Obfuscator (Canada)

И последней библиотекой для обфускации кода является ESMangle, которая состоит в пакете Esprima и может использовать в виде библиотеки JavaScript или пакета для Node.js. Эта библиотека не производит сложных преобразований исходного кода. Она делает переименование переменных, минимизацию и небольшую оптимизацию.

Изучение обфускаторов под платформу .NET

Перед началом выполнения работы необходимо скачать и установить следующие утилиты:

1. Telerik JustDecompile <http://www.telerik.com/products/decompiler.aspx>
2. Eazfuscator.NET <http://eazfuscator-net.software.informer.com/3.3/>

При помощи утилиты JustDecompile необходимо открыть exe- или dll-файл, скомпилированный под платформу .NET. В левой панели можно просматривать структуру содержимого файла: классы и методы, при выборе метода в правой панели отображается содержимое соответствующего метода (рис.3).

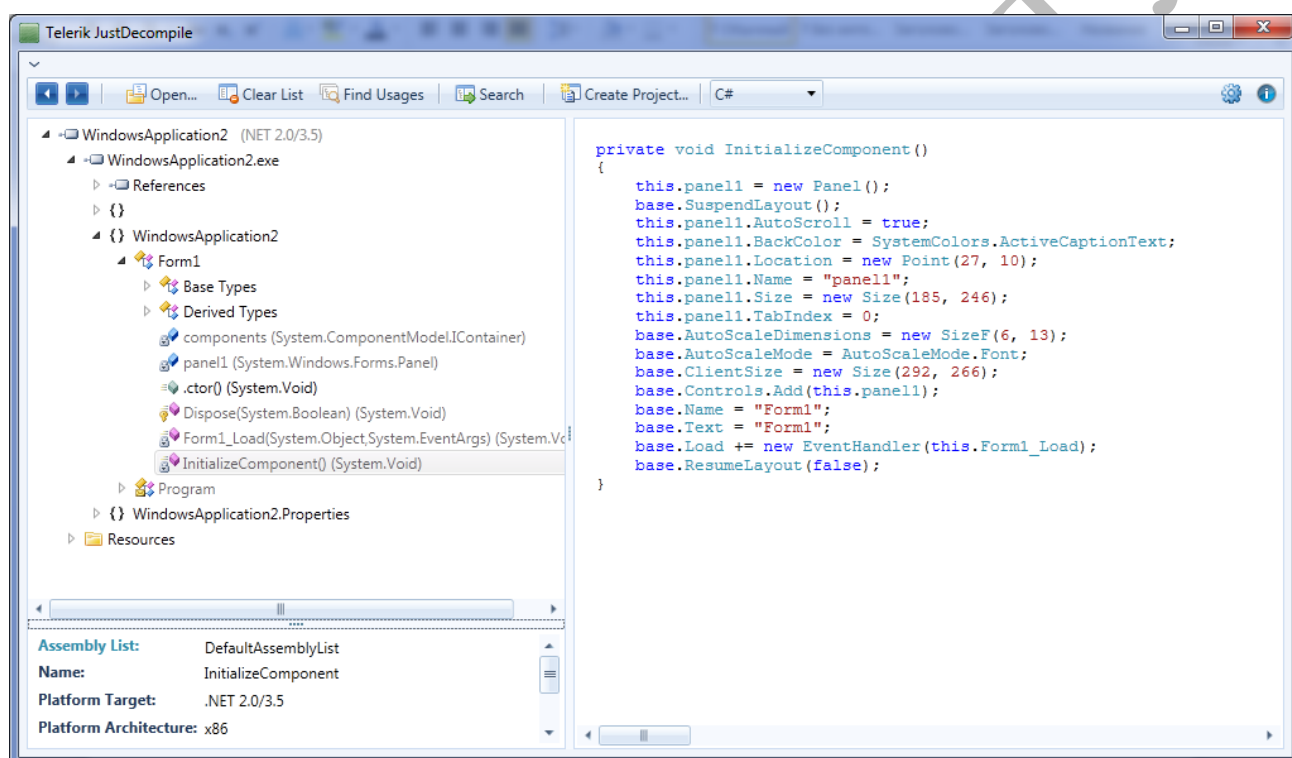


Рисунок 3 – Содержимое файла WindowsApplication2.exe до обфускации

Далее необходимо открыть проект в VisualStudio, для которого будет выполняться обфускация, и запустить утилиту Eazfuscator.NET Assistant. Для того чтобы включить режим обфускации для проекта, достаточно перетянуть проект из панели Solution Explorer в Visual Studio на зеленую зону в Eazfuscator. NET Assistant (рис.4). После этого при каждой компиляции программы в режиме Release будет выполняться обфускация.

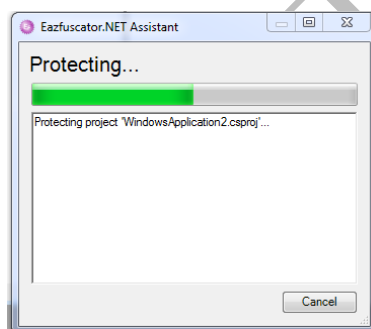
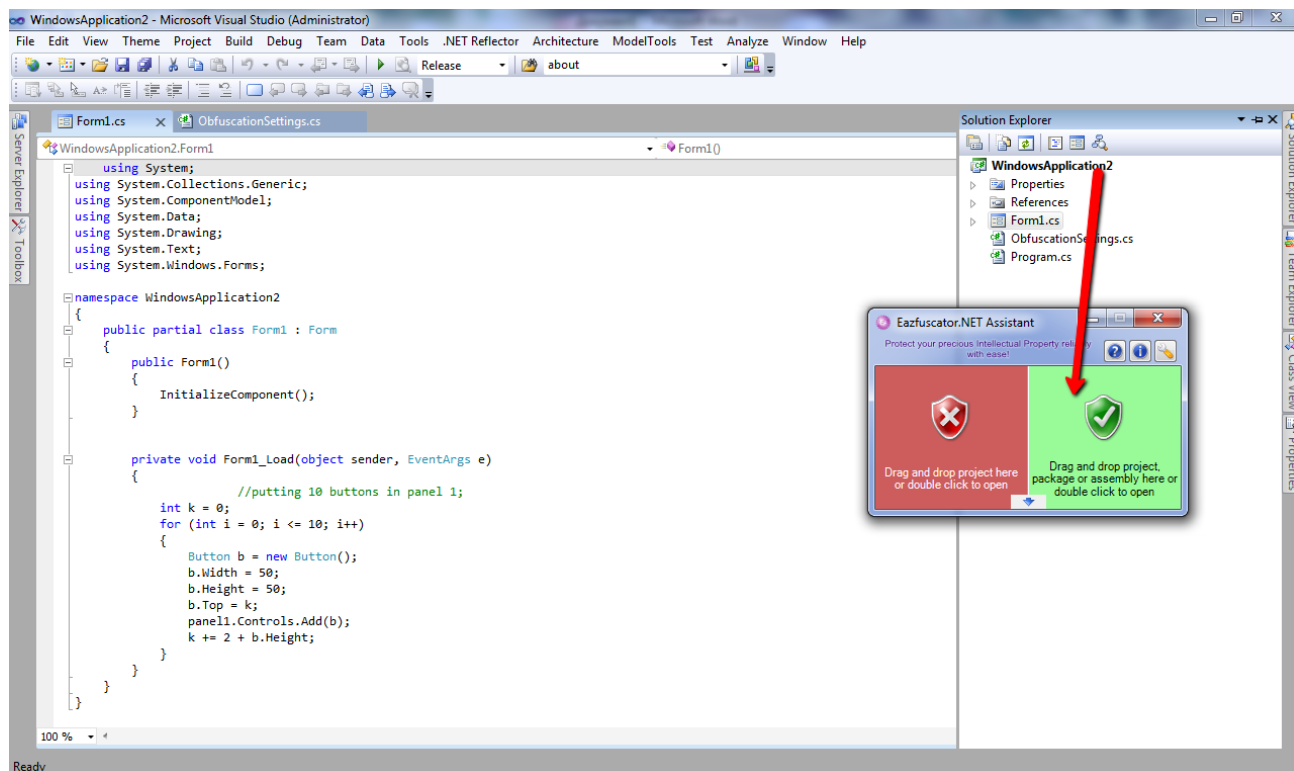


Рисунок 4 – Включение режима обфускации для проекта Visual Studio

Для настройки обфускации необходимо добавить файл ObfuscationSettings.cs в проект. Например, чтобы шифровать строковые константы содержимое файла ObfuscationSettings.cs должно быть следующим:

```
using System;
using System.Reflection;
```

```
[assembly: Obfuscation(Feature = "encrypt symbol names with password XXXXXX", Exclude = false)]
```

После этого необходимо выполнить компиляцию программы в Visual Studio в режиме Release. В окне Output в Visual Studio будет видно, что была выполнена обфускация файла WindowsApplication2.exe (рис.5).

```

----- Build started: Project: WindowsApplication2, Configuration: Release Any CPU -----
WindowsApplication2 -> C:\WindowsApplication2\bin\Release\WindowsApplication2.exe
Obfuscating assembly 'WindowsApplication2.exe'...
Done
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====

```

Рисунок 5 – Обфускация при компиляции проекта в режиме Release

Далее снова запускаем утилиту JustDecompile и просматриваем результаты обфускации. Как видно на рис.6 названия классов, названия и содержимое методов были заменены на случайные символы.

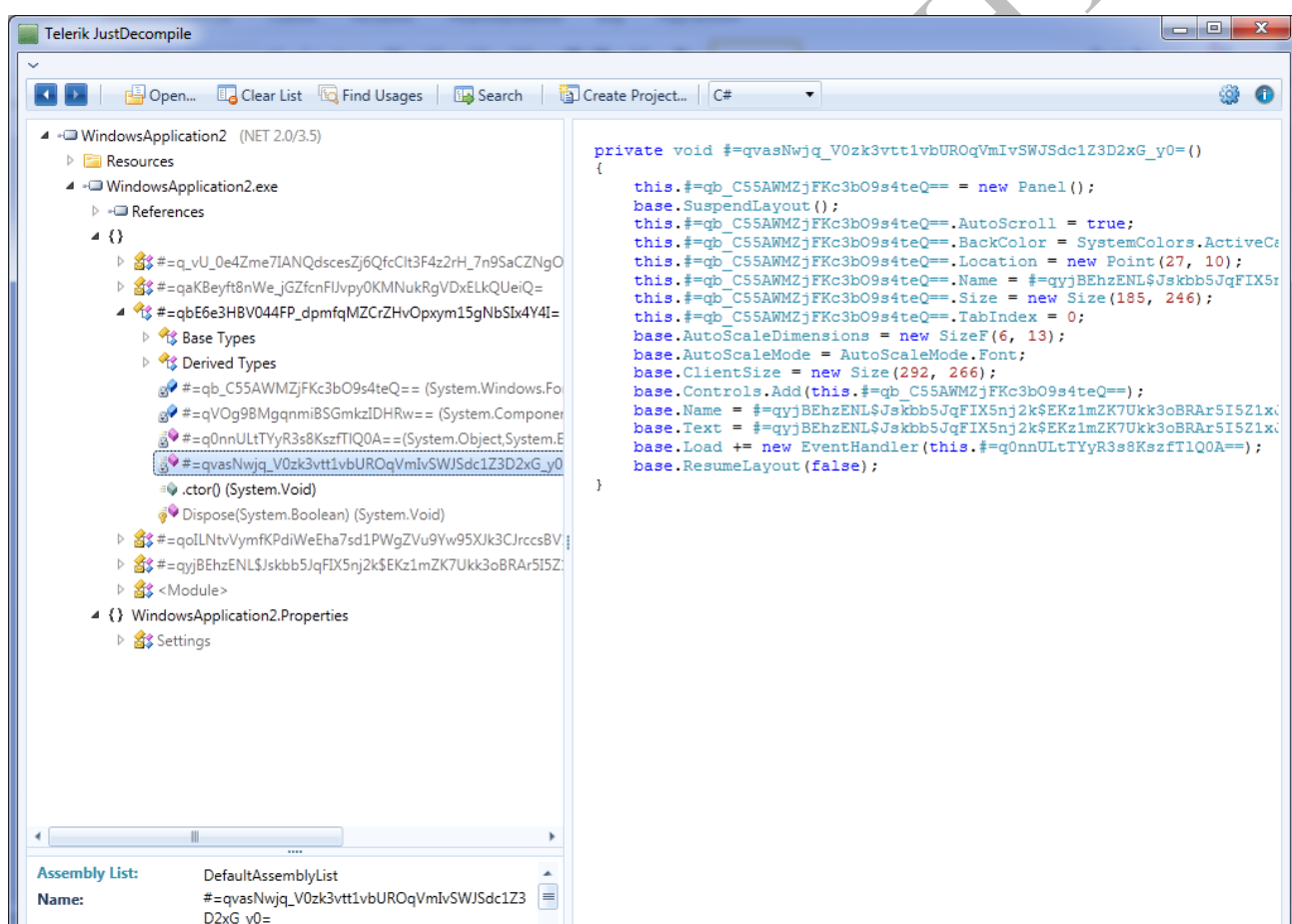


Рисунок 6 – Содержимое файла WindowsApplication.exe после обфускации

Задание к лабораторной работе:

- 1) Изучить теоретические сведения по предложенной теме.

- 2) Изучить в рамках собственного проекта возможности представленных в лабораторной работе утилит для обфускации, отметить достоинства и недостатки.
- 3) Использовать существующие (предложенные) средства обфускации для защиты авторства программного кода.

Контрольные вопросы:

- 1) Что такое обфускация?
- 2) Какие виды обфускации Вы знаете?
- 3) Всегда ли процесс деобфускации приводит к первоначальному коду?

Кафедра ПИ ДонНТУ

Литература

1. А.Ю. Щеглов «Защита компьютерной информации от несанкционированного доступа», -Наука и техника, Ст.Петербург, 2004 г., 384 стр.
2. В.И. Загородний, «Комплексная защита информации в компьютерных системах», Москва, Логос, 2001 г. – 266 стр.
3. Ю.В. Романец, П.А. Тимофеев, В.Ф.Шаньгин, «Защита информации в компьютерных системах и сетях», Москва, Радио и связь - 2001 г. – 190 стр.
4. Стивен Норткатт, Джуди Новак, Обнаружение вторжений в сеть, Лори, 2001 г. – 384 стр.
5. Вильям Столлинс, Криптография и защита сетей, Вильямс, 2001 г., - 669 стр.
6. Соколов А.В., Шаньгин В.Ф., Защита информации в распределенных корпоративных сетях и системах, ДМК, 2002 г. – 655 стр.
7. Д. Бэндел, Защита и безопасность в сетях Linux., 324 стр.
8. А.Кенин, Самоучитель системного администратора, БХВ, 2008 г., 536 стр.
9. W indows 2003 server, Справочное руководство.
10. Баричев С. Криптография без секретов, эл. вариант.
11. Ресурсы сети Internet: <http://citforum.ru/security/>
12. Теория информационной безопасности и методология защиты информации [Электронный ресурс] // ifmo – Режим доступа: <http://books.ifmo.ru/file/pdf/704.pdf>
13. Информатика: введение в информационную безопасность [Электронный ресурс] // rumvi – Режим доступа: <http://www.rumvi.com/products/ebook/информатика-введение-в-информационную-безопасность/4e6d02d0-1f0d-4e1a-930a-2a452ced9d56/preview/preview.html>

14. Введение в криптографию [Электронный ресурс] // citforum – Режим доступа: http://citforum.ck.ua/security/cryptography/crypto_1.shtml
15. Основные понятия, термины и определения компьютерной стеганографии [Электронный ресурс] // scipeople – Режим доступа: <http://scipeople.ru/group/4720/topic/6430/>
16. А.В. Аграновский, А.В. Балакин, В.Г. Грибунин, С.А. Сапожников. Стеганография, цифровые водяные знаки и стегоанализ. – М.: Вузовская книга, 2009. – 220 с.
17. Стеганография в XXI веке. Цели. Практическое применение. Актуальность [Электронный ресурс] // Хабрахабр – Режим доступа: <https://habrahabr.ru/post/253045/>
18. Г.Ф. Конахович, А. Ю. Пузыренко. Компьютерная стеганография. Теория и практика – К.: МК-Пресс, 2006. – 288 с.
19. DarkJPEG: стеганография для всех [Электронный ресурс] // Хабрахабр – Режим доступа: <https://habrahabr.ru/post/187402/>
20. JPEG steganography web service [Электронный ресурс] // Github – Режим доступа: <https://github.com/yndi/darkjpeg>
21. OpenStego [Электронный ресурс] // Github – Режим доступа: <https://github.com/syvaitya/openstego>
22. LSB-Steganography [Электронный ресурс] // Github – Режим доступа: <https://github.com/RobinDavid/LSB-Steganography>
23. Дж. Миано. Форматы и алгоритмы сжатия изображений в действии М.: Издательство Триумф, 2003 – 336 с.
24. Википедия. Обфускация [Электронный ресурс]. – URL: <https://ru.wikipedia.org/wiki/Обфускация>
25. JavaScriptObfuscator. Canada [Электронный ресурс]. – URL: <https://www.javascriptobfuscator.com/>
26. Введение в JSON [Электронный ресурс]. – URL: <http://www.json.org/json-ru.html>

27.ParserAPI [Электронный ресурс]. – URL:
https://developer.mozilla.org/enUS/docs/Mozilla/Projects/SpiderMonkey/Parser_API

Кафедра ПИ ДонНТУ

СОДЕРЖАНИЕ

Лабораторная работа №1 Изучение возможностей программных закладок, логических «бомб» и «троянских коней.

Лабораторная работа №2. Базовые алгоритмы шифрования.

Использование симметричного алгоритма «Сеть Фейстеля» для шифрования файлов на диске.

Лабораторная работа № 3. Средства защиты ОС семейства Windows.

Лабораторная работа №4 Изучение стандартных средств для реализации приложений, использующих симметричное и асимметричное шифрование.

Лабораторная работа №5 Защита автоматизированной системы

Лабораторная работа №6 Использование стеганографических и криптографических средств защиты информации.

Лабораторная работа №7 «Обфускация программ»

Литература

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ
к лабораторным работам по дисциплине
«Безопасность программ и данных»

(для студентов направления подготовки 09.03.04 “Программная инженерия ”)

Составители:

Алла Викторовна Чернышова