**Лекция 12. Основные подходы к защите программ от несанкционированного копирования. Краткое описание криптографических средств контроля целостности и достоверности программ**

**Основные функции средств защиты от копирования**

При защите программ от несанкционированного копирования применяются методы, которые позволяют привносить в защищаемую программу функции привязки процесса выполнения кода программы только на тех ЭВМ, на которые они были инсталлированы.

Инсталлированная программа для защиты от копирования при каждом запуске должна выполнять следующие действия:

* анализ аппаратно-программной среды компьютера, на котором она запущена, формирование на основе этого анализа текущих характеристик своей среды выполнения;
* проверка подлинности среды выполнения путем сравнения ее текущих характеристик с эталонными, хранящимися на винчестере;
* блокирование дальнейшей работы программы при несовпадении текущих характеристик с эталонными.

Этап проверки подлинности среды является одним из самых уязвимых с точки зрения защиты. Можно детально не разбираться с логикой защиты, а немного «подправить» результат сравнения, и защита будет снята.

При выполнении процесса проверки подлинности среды возможны три варианта: с использованием множества операторов сравнения того, что есть, с тем, что должно быть, с использованием механизма генерации исполняемых команд в зависимости от результатов работы защитного механизма и с использованием арифметических операций. При использовании механизма генерации исполняемых команд в первом байте хранится исходная ключевая контрольная сумма BIOS, во второй байт записывается подсчитанная контрольная сумма в процессе выполнения задачи. Затем осуществляется вычитание из значения первого байта значение второго байта, а полученный результат добавляется к каждой ячейки оперативной памяти в области операционной системы. Понятно, что если суммы не совпадут, то операционная система функционировать не будет. При использовании арифметических операций осуществляется преобразование над данными арифметического характера в зависимости от результатов работы защитного механизма.

Для снятия защиты от копирования применяют два основных метода: статический и динамический.

Статические методы предусматривают анализ текстов защищенных программ в естественном или преобразованном виде. Динамические методы предусматривают слежение за выполнением программы с помощью специальных средств снятия защиты от копирования.

**Основные методы защиты от копирования**

***1. Криптографические методы***

Для защиты инсталлируемой программы от копирования при помощи криптографических методов инсталлятор программы должен выполнить следующие функции:

* анализ аппаратно-программной среды компьютера, на котором должна будет выполняться инсталлируемая программа, и формирование на основе этого анализа эталонных характеристик среды выполнения программы;
* запись криптографически преобразованных эталонных характеристик аппаратно-программной среды компьютера на винчестер.

Преобразованные эталонные характеристики аппаратно-программной среды могут быть занесены в следующие области жесткого диска:

* в любые места области данных (в созданный для этого отдельный файл, в отдельные кластеры, которые должны помечаться затем в FAT как зарезервированные под операционную систему или дефектные);
* в зарезервированные сектора системной области винчестера;
* непосредственно в файлы размещения защищаемой программной системы, например, в файл настройки ее параметров функционирования.

Можно выделить два основных метода защиты от копирования с  использованием криптографических приемов:

* + с использованием односторонней функции;
	+ с использованием шифрования (которое также может использовать  односторонние функции).

Односторонние функции это функции, для которых при любом *x* из  области определения легко вычислить *f*(*x*), однако почти для всех *y* из ее области значений, найти *y*=*f*(*x*) вычислительно трудно.

Если эталонные характеристики программно-аппаратной среды представить в виде аргумента односторонней функции *x*, то *y* - есть «образ» этих характеристик, который хранится на винчестере и по значению которого вычислительно невозможно получить сами характеристики. Примером такой односторонней функции может служить функция дискретного экспоненцирования с размерностью операндов не менее 512 битов.

При шифровании эталонные характеристики шифруются по ключу, совпадающему с этими текущими характеристиками, а текущие характеристики среды выполнения программы для сравнения с эталонными также зашифровываются, но по ключу, совпадающему с этими текущими характеристиками. Таким образом, при сравнении эталонные и текущие характеристики находятся в зашифрованном виде и будут совпадать только в том случае, если исходные эталонные характеристики совпадают с исходными текущими.

***2. Метод привязки к идентификатору***

В случае если характеристики аппаратно-программной среды отсутствуют в явном виде или их определение значительно замедляет запуск программ или снижает удобство их использования, то для защиты программ от несанкционированного копирования можно использовать методы привязки к идентификатору, формируемому инсталлятором. Суть данного метода заключается в том, что на винчестере при инсталляции защищаемой от копирования программы формируется уникальный идентификатор, наличие которого затем проверяется инсталлированной программой при каждом ее запуске. При отсутствии или несовпадении этого идентификатора программа блокирует свое дальнейшее выполнение.

Основным требованием к записанному на винчестер уникальному идентификатору является требование, согласно которому данный идентификатор не должен копироваться стандартным способом. Для этого идентификатор целесообразно записывать в следующие области жесткого диска:

• в отдельные кластеры области данных, которые должны помечаться затем в FAT как зарезервированные под операционную систему или как дефектные;

• в зарезервированные сектора системной области винчестера.

Некопируемый стандартным образом идентификатор может помещаться на дискету, к которой должна будет обращаться при каждом своем запуске программа. Такую дискету называют ключевой. Кроме того, защищаемая от копирования программа может быть привязана и к уникальным характеристикам ключевой дискеты. Следует учитывать, что при использовании ключевой дискеты значительно увеличивается неудобство пользователя, так как он всегда должен вставлять в дисковод эту дискету перед запуском защищаемой от копирования программы.

***3. Методы, основанные на работе с переходами и стеком***

Данные методы основаны на включение в тело программы переходов по динамически изменяемым адресам и прерываниям, а также самогенерирующихся команд (например, команд, полученных с помощью сложения и вычитания). Кроме того, вместо команды безусловного перехода может использоваться возврат из подпрограммы. Предварительно в стек записывается адрес перехода, который в процессе работы программы модифицируется непосредственно в стеке.

При работе со стеком, стек определяется непосредственно в области исполняемых команд, что приводит к затиранию при работе со стеком. Этот способ применяется, когда не требуется повторное исполнение кода программы. Таким же способом можно генерировать исполняемые команды до начала вычислительного процесса.

***4. Манипуляции с кодом программы***

При манипуляциях с кодом программы можно привести два следующих способа:

• включение в тело программы «пустых» модулей;

• изменение защищаемой программы.

Первый способ заключается во включении в тело программы модулей, на которые имитируется передача управления, но реально никогда не осуществляется. Эти модули содержат большое количество команд, не имеющих никакого отношения к логике работы программы. Но «ненужность» этих программ не должна быть очевидна потенциальному злоумышленнику.

Второй способ заключается в изменении начала защищаемой программы таким образом, чтобы стандартный дизассемблер не смог ее правильно дизассемблировать. Например, такие программы, как Nota и Copylock, внедряя защитный механизм в защищаемый файл, полностью модифицируют исходный заголовок EXE-файла.

Все перечисленные методы были, в основном направлены на противодействия статическим способам снятия защиты от копирования. В следующем подразделе рассмотрим методы противодействия динамическим способам снятия защиты.

***5. Методы противодействия динамическим способам снятия защиты программ от копирования***

Набор методов противодействия динамическим способам снятия защиты программ от копирования включает следующие методы:

• периодический подсчет контрольной суммы, занимаемой образом задачи области оперативной памяти, в процессе выполнения. Это позволяет:

- заметить изменения, внесенные в загрузочный модуль;

- в случае если программу пытаются «раздеть», выявить контрольные точки, установленные отладчиком;

* проверка количества свободной памяти и сравнение и с тем объемом, к которому задача «привыкла» или «приучена». Это действия позволит застраховаться от слишком грубой слежки за программой с помощью резидентных модулей;
* проверка содержимого незадействованных для решения  защищаемой программы областей памяти, которые не попадают под общее распределение оперативной памяти, доступной для программиста, что позволяет добиться «монопольного» режима работы программы;
* проверка содержимого векторов прерываний (особенно 13h и 21h) на наличие тех значений, к которым задача «приучена». Иногда бывает полезным сравнение первых команд операционной системы, отрабатывающих этим прерывания, с теми командами, которые там должны быть. Вместе с предварительной очисткой оперативной памяти проверка векторов прерываний и их принудительное восстановление позволяет избавиться от большинства присутствующих в памяти резидентных программ;
* переустановка векторов прерываний. Содержимое некоторых векторов прерываний (например, 13h и 21h) копируется в область свободных векторов. Соответственно изменяются и обращения к прерываниям. При этом слежение за известными векторами не даст желаемого результата. Например, самыми первыми исполняемыми командами программы копируется содержимое вектора 21h (4 байта) в вектор 60h, а вместо команд int 21h в программе везде записывается команда int 60h. В результате в явном виде в тексте программы нет ни одной команды работы с прерыванием 21h;
* постоянное чередование команд разрешения и запрещения прерывания, что затрудняет установку отладчиком контрольных точек;
* контроль времени выполнения отдельных частей программы, что позволяет выявить «остановы» в теле исполняемого модуля.

Многие перечисленные защитные средства могут быть реализованы исключительно на языке Ассемблер. Одна из основных отличительных особенностей этого языка заключается в том, что для него не существует ограничений в области работы со стеком, регистрами, памятью, портами ввода/вывода и т.п.

**Криптопрограммирование посредством использования инкрементальных алгоритмов**

Одним из основных инструментов методологии криптопрограммирования являются *инкрементальные криптографические алгоритмы*. Цель *инкрементальной криптографии* заключается в разработке криптографических алгоритмов обработки электронных данных, обладающих следующим принципиальным свойством. Если алгоритм применяется к электронным данным *D* для достижения каких- либо их защитных свойств, то применение инкрементального алгоритма к данным *D*, подвергнувшихся модификации – *D*′, должно осуществляться быстрее, чем необходимость заново обработать первоначальный электронный документ. В тех приложениях, когда указанные алгоритмы используют, например, алгоритмы шифрования электронных документов или их цифровой подписи, требование повышения эффективности инкрементальных алгоритмов является основным. Один из основных методов применения инкрементальных алгоритмов заключается в использовании их аутентификационных признаков для антивирусной защиты.

При обработке электронных документов инкрементальными алгоритмами рассматриваются такие операции обработки данных как «вставка» и «стирание» для символьных строк или «cut» - «вырезание и помещение в буфер» и «paste» - «извлечение из буфера и вставка» для текста. Основная задача здесь заключается в разработке эффективных инкрементальных алгоритмов для схем цифровой подписи и схем аутентификации сообщений, поддерживающих вышеупомянутые операции по модификации электронных данных. Такие алгоритмы должны обладать основным качественным свойством, а именно *свойством защиты от вмешательства*, что, таким образом, и делает их применимыми для *защиты программ от вирусов и других разрушающих программных средств*.

Основные криптографические примитивы, такие как шифрование и цифровая подпись имеют фундаментальную теоретическую базу. Во многих работах были даны базовые определения их криптографической стойкости, основанные на обобщенных теоретико-сложностных и теоретико-информационных предположениях. Главная проблема, которая остается и затрудняет использование на практике многих доказуемо стойких теоретических криптоконструкций, заключается в их пространственно-временной неэффективности. Инкрементальность, в этом смысле, является новой мерой эффективности, которая является вполне приемлемой во многих алгоритмических приложениях.

Пусть далее рассматривается процессор, защищенный от физического вмешательства, который имеет ограниченное количество безопасной локальной памяти. Необходимо получить доступ к файлам, находящихся на удаленных (возможно небезопасных) носителях, например, хост- станциях или *www*-серверах. Компьютерный вирус может атаковать удаленную станцию, и исследовать и изменять содержание удаленной информационной среды (но при этом он не имеет доступа к безопасной локальной памяти процессора). Для защиты файлов от таких вирусов, процессор вычисляет для каждого файла аутентификационный признак, как функцию от самого файла и ключа, который хранится в безопасной локальной памяти.

Такая организация защиты при внедрении вируса в файл не позволит вирусу вычислять (или получить каким-либо «известным только вирусу» образом) новый аутентификационный признак, а значит при реализации процесса верификации признака, таким образом, обнаружится вторжение вируса. Следует обратить внимание на то, что для корректной верификации аутентификационного признака защищенный процессор должен заново подтвердить подлинность файлов. Очевидно, что наиболее привлекательным способом такой организации защиты от вирусов является модернизация аутентификационного признака быстрее, чем необходимость его вычисления заново. Эта проблема особенно сложна в том случае, когда защищенная локальная память является не достаточно большой для того, чтобы хранить (даже временно) фрагмент файла или когда «слишком ресурсозатратно» ввести в локальную память полный файл.

Таким образом, основная идея инкрементальных алгоритмов, состоит в том, чтобы воспользоваться какими-либо имеющимися преимуществами организации программно-аппаратного процесса вычислений и найти такие способы криптографических преобразований над электронными данными *D* которые позволят обрабатывать (в целях их защиты) всякий раз эти данные не заново, а обрабатывать (посредством быстродействующих криптографических преобразований) уже имеющиеся аутентификационные признаки, которые ранее были получены для *D*. Когда «изменения» в обрабатываемых электронных данных не велики, инкрементальные методы могут дать большие преимущества по эффективности.

**Основные элементы инкрементальной криптографии**

***Базовые примитивы***

Инкрементальность можно рассматривать для любого криптографического примитива. В данном случае рассматриваются два из них – цифровая подпись и шифрование. Инкрементальность далее рассматривается, как правило, для «прямых» преобразований, а именно для генерации подписи и шифрования, но все рассуждения будут верны и для «сопряженных» преобразований, а именно для верификации подписи и дешифрования.

***Операции модификации***

При рассмотрении проблемы модификации защищаемого файла в терминах применения фиксированного набора основных операций по модификации электронного документа исследуются следующие операции модификации: замена блока в документе другим; вставка нового блока; удаление старого блока. Операции должны быть достаточно «мощны» для демонстрации реальных модификаций таких как: замена, вставка и удаление. Соответственно также рассматриваются операции «cut» и «paste», например, операции разбиения отдельного документа на два, а затем, вставка двух документов в один.

***Инкрементальные алгоритмы***

Зафиксируем базовое криптографическое преобразование **T** (например, цифровая подпись документа с некоторым ключом). Каждой элементарной операции модификации текста (например, вставки) будет соответствовать инкрементальный алгоритм **I**. На вход этого алгоритма подаются: исходный файл, значения преобразования **T** на нем, описание операций модификации и, возможно, соответствующие ключи или другие параметры. Это позволяет вычислить значение **T** для результирующего файла. Основная проблема здесь заключается в проектировании схем обработки файлов, с включенными в них эффективными инкрементальными алгоритмами. Предположим, что имеется подпись σстар для файла *D*стар и файл *D*′стар, измененный посредством вставки в файл *D*стар некоторых данных. Необходимо получить новую цифровую подпись путем подписывания строки, состоящей из σстар и описания операций модификации над документом *D*стар. Это схема называется *схемой, зависящей от истории*. Могут иметься приложения, когда такие действия могут применяться. В большинстве же случаев это не желательно, так как когда делается большое количество изменений, то затраты на верификацию подписи (а эти затраты пропорциональны числу изменений) резко увеличиваются. В связи с этим размеры подписи растут со временем. Чтобы избежать таких затрат необходимо использовать *схемы, свободные от истории* или *HF*-*схемы*. Все нижеприведенные схемы являются схемами, свободными от истории.

***Безопасность***

Свойство инкрементальности вводит новые проблемы безопасности, а, следовательно, «назревает» необходимость новых определений. Рассмотрим случай схем подписи или аутентификации сообщений. Разумно предположить, что противник не только имеет доступ к предыдущим подписанным версиям файлов, но также способен выдавать команды на модификацию текста в существующих файлах и получать инкрементальные подписи для измененных файлов. Такая атака на основе выбранного сообщения для инкрементальных алгоритмов подписи может вести к «взлому» используемой оригинальной схемы подписи, которая не может быть взломана при проведении противником атак, когда инкрементальные алгоритмы не используются. Кроме того, в некоторых сценариях, например, при вирусных атаках можно предположить, что противник может вмешиваться не только в содержание существующих документов, но и в соответствующие аутентификационные признаки, полученные посредством применения схемы подписи (или схемы аутентификации сообщений). Соответственно рассматриваются два определения безопасности: базовое, когда необходимо противостоять первому вышеописанному противнику, и более сильное понятие безопасности, когда доказывается стойкость защиты от вмешательств.

***Секретность в инкрементальных схемах***

Исходя из вышесказанного, появляется новая проблема, которая проявляется в инкрементальном сценарии, а именно - проблема секретности различных версий файлов. Предположим μ - подпись для электронных данных *М* и μ*'* является подписью несколько измененных данных *M'*. Тогда, нам необходимо построить такую инкрементальную схему получения подписи μ*'*, в которой последняя (подпись μ*'*) давала бы как можно меньше информации об оригинальном коде *М*.

**Методы защиты данных посредством инкрементальных алгоритмов маркирования**

***Инкрементальная аутентификация***

*Основные определения и обозначения.* Пусть **АУТ**(*m*) - обычный (оригинальный) алгоритм аутентификации сообщений и **АУТ**α(*m*)- функция маркирования сообщения *m*, индуцированная схемой **АУТ** с ключом аутентификации α. Пусть **ВЕР**α(*m*,β) - соответствующий алгоритм верификации, где β={true, false} – предикат корректности проверки.

Далее будут использоваться деревья поиска и, следовательно, необходимо напомнить, что 2-3-дерево имеет все концевые узлы (листья) на одном и том же самом уровне/высоте (как и в случае сбалансированных двоичных деревьев) и каждая внутренняя вершина имеет или 2, или 3 дочерних узла. В данном случае 2-3-дерево подобно двоичному дереву является упорядоченным деревом и, таким образом, концевые узлы являются упорядоченными. Пусть *Vh* – определяет множество всех строк длины не больше *h*, ассоциированных очевидным образом с вершинами сбалансированного 2-3-дерева высоты *h*. Маркированное дерево может рассматриваться как функция *Т*: *Vh*→{0,1}\*, которая приписывает аутентификационный признак (АП) каждой вершине.

Пусть совокупный аутентификационный признак файла *F* получен посредством использования 2-3-дерева аутентификационных признаков для каждого из блоков файла *F*=*F*[1],...,*F*[*l*] (далее такое дерево будет называться *маркированным деревом*). Каждая вершина *w* ассоциирована с меткой, которая состоит из АП (аутентифицирующих дочерние узлы) и счетчика, представляющего число узлов в поддереве с корнем *w*.

*Алгоритм маркирования*

Алгоритм создания 2-3-дерева аутентификационных признаков (алгоритм маркирования) работает следующим образом.

**Алгоритм САП2-3**

1. Получить для каждого *i*, признак (α,*F*[*i*])**АУТ=(***T*(*w*)), где *w* – *i*-тый концевой узел.

2.Получить для каждого неконцевого узла *w*, признак

(α,(*L*1,*L*2,*L*3),*рзм*)**АУТ=(***Т*(*w*)), где *Li* - метка *i*-того дочернего узла *w* (в случае, если *w* имеет только два дочерних узла, то *L*3=γ) и *рзм* - число узлов в поддереве с корнем *w*).

3. Получить для корня дерева признак (α,(*L*1,*L*2,*L*3),*Id*,*счт*)**АУТ=(***Т*(λ)), где *Id* - название документа и *счт* – соответствующее показание счетчика (связанное с этим документом).

*Инкрементальный алгоритм маркирования.* Предположим, что файл *F*, аутентифицированный маркированным деревом, подвергается операции замены, то есть *j*-тый блок файла *F* заменен блоком *F*(σ). Сначала необходимо проверить, что путь от требуемого текущего значения до корня дерева корректен. Для этого необходимо выполнить следующий алгоритм.

**Алгоритм ИАМ2-3**

Пусть *u*0,...*uh* - путь из корня *u*0=λ к *j*-тому концевому узлу обозначается как *uh*. Тогда:

1. Проверить, что алгоритм **ВЕР**α при верификации принимает *Т*(λ) как корректный АП строки (α,(*L*1,*L*2,*L*3),*Id*,*счт*)**АУТ=***Т*(λ), где *Id* - название документа и *счт* - текущее значение счетчика (связанного с этим документом).

2. Для *i*=1,...,*h*-1 проверить, что **ВЕР**α принимает *Т*(*ui*) как корректный АП строки ((*L*1,*L*2,*L*3),*рзм*), где *Li* - метка *i*-того дочернего узла *w* (в случае, если *w* имеет только два дочерних узла, то *L*3=γ) и *рзм* - число узлов в поддереве с корнем *w*)).

3. Проверить, что **ВЕР**α принимает *Т*(*uh*) как корректный АП блока *F*[*j*].

4. Если все эти проверки успешны, тогда совокупный АП файла *F* получается следующим образом.

4.1. Установить *Т*(*uh*):=АУТ(*F*(σ*)*).

4.2. Для *i*=*h*-1,...,1 установить *Т*(*ui*):=АУТ(*Т*(*ui*1),*Т*(*ui*2),*Т*(*ui*3)).

4.3. Установить *Т*(λ):=АУТ((*Т*(*ui*0),*Т*(*ui*1),*Т*(*ui*1)),*Id*,*счт*+1).

Следует подчеркнуть, что значения *Т* на всех других вершинах (то есть, не стоящих на пути *u*0,...,*uh*) остаются неизменяемыми.

Следует также отметить, что предлагаемая инкрементальная схема маркирования имеет дополнительное свойство, заключающееся в том, что она безопасна даже для противника, который может «видеть» как отдельные аутентификационные признаки, так и все маркированное дерево и может даже «вмешиваться» в эти признаки. Для каждого файла, пользователь должен хранить в локальной безопасной памяти ключ *x* схемы подписи, имя файла и текущее значение счетчика. Всякий раз, когда пользователь хочет проверить целостность файла, он проверяет корректность маркированного дерева открытым образом.

Наиболее эффективным является использование инкрементального алгоритма маркирования для защиты программ, использующих постоянно обновляющие структуры данных, например, файл с исходными данными для программ или итерационно изменяемыми переменными.

***Инкрементальное шифрование***

*Вводные замечания.* Будем говорить, что инкрементальное шифрование является *стойким относительно некоторой операции модификации*, если (для данной последовательности шифртекстов *Е*1,...,*Et* над соответствующими открытыми данными *Di*, *i*=1,...,*t* и при инкрементальном получении каждой последовательности *Еi*, из предыдущих шифртекстов *Ei*-1) извлечение какой-либо информации об оригинальном документе *D*1, также как и его измененных версиях *D*2,.., *Dt* (за исключением того факта, что *Di* получено посредством применения этой операции модификации к открытым данным*Di*-1) *не возможно*. Аналогично, рассмотрим любые две последовательности *А*=(*А*1,...,*Аt*) и *В*=(*В*1,...,*B*B*t*) так, что *A*1,*B*B1∈Σ*l*, где Σ - используемый алфавит. Данные *Аi* (отн., *Вi*) получены заменой единственного символа в *Ai*-1 (отн., *Вi*-1). Тогда, не должны быть различимы последовательность шифртекстов, полученных посредством применения инкрементального алгоритма *I* при обработке «командой создания» блока данных *A*1 и соответствующих «команд замены» в последовательности *А*, от последовательности шифртекстов, полученных посредством применения инкрементального алгоритма *I* при обработке «команды создания» блока данных *В*1 и соответствующих «команд замены» данных *В*.

*Схемы инкрементального шифрования*

Предлагаемые решения используют стойкую схему вероятностного шифрования *E*. Предположим, что *Е* может использоваться для шифрования символов из Σ и пар (*i*,δ), где δ∈Σ и *i* - целое число не большее, чем длина блоков данных в системе. При использовании *Е* сначала будет описан алгоритм, который шифрует версии блоков данных, в которых осуществляется операции замены. Далее будем рассматривать операцию замены одного символа. Шифрованные версии состоят из двух последовательностей шифртекстов, обозначаемых *Е*1 и *Е*2. Первая последовательность *Е*1 является результатом блочного шифрования некоторого файла *D*=*D*[1]...*D*[*l*], в то время как вторая *Е*2, шифрует последовательность изменений, обозначаемую *M*=*М*[1]...*M*[*t*] , посредством которых текущий документ был получен из *D*. Тогда инкрементальный алгоритм шифрования заключается в конкатенации шифртекста модифицированного документа с *Е*2. Каждые *l* шагов алгоритм восстанавливает модифицированные данные и заново шифрует их, используя блочное шифрование и, таким образом, формирует новую последовательность шифртекстов *Е*1 (одновременно устанавливая *Е*2 в нулевую последовательность). Сложность этого алгоритма составляет два блочных шифрования на каждое изменение. Важным моментом является то, что конвейерная обработка является достаточно ресурсозатратной.

Теперь предположим, что нам позволено сохранять промежуточные результаты работы в некоторой безопасной памяти («невидимой противником»). В этом случае, как только длина *Е*2 достигнет *l*, мы можем начать подготовку новых шифртекстов для данных, обозначаемых *D'*, которые получаются *D* применением первых *l* изменений в *М*. Для этого необходимо выполнить все требуемые вычисления наряду со следующими *l* изменениями, в то время как *М* увеличивается до размера 2*l*. Таким образом, мы имеем шифртекст, обозначаемый *Е'* данных *D'* (конечно, теперь текущие данные другие). При замене *Е*1 на *Е'* и при исключении первых *l* изменений в *М* мы получаем зашифрованную форму текущего документа. Следует подчеркнуть, что в такой модели вычислений все эти операции могут выполняться за константное время. Наконец необходимо отметить, что алгоритм работает в *периодах*, каждый состоящий из *l* изменений. В каждом периоде, алгоритм модифицирует шифртекст для сравнения с изменениями, выполненными в предыдущем периоде.

Выше мы предположили, что пользователь может хранить промежуточные результаты (которые требуют памяти размером *О*(*l*)) в локальной памяти, невидимой противником. Это предположение нереалистично в некоторых сценариях и несовместимо с вышеприведенными определениями для схем инкрементального шифрования. Далее мы используем вышеупомянутые идеи без этого предположения. Для этого нам необходимо шифровать данные, используя три последовательности, обозначаемые *Е*1,*Е*2 и *Е*3. Первые две последовательности *Е*1 и *Е*2, является точно такими же, как определенные выше. Их достаточно для дешифрования данных. Дополнительная последовательность *Е*3 – является «шифрованием рабочей области» и обозначается как *W*=*W*[1]...*W*[2*l*].

Далее мы опишем операции, выполняемые в одном периоде. В нашем описании, мы не упоминаем явно операции шифрования, и, таким образом, всякий раз, когда мы говорим, что мы устанавливаем элемент последовательности *W*, это должно означать, что соответствующий шифртекст получен и сохранен.

Сначала, мы устанавливаем элементы последовательности *W* следующим образом: *W*[*i*]:=(*i*+*D*[*i*]) для *i*≤*l* и *W*[*i*]:=*M*[*i*-*l*] для *l*+1≤*i*≤2*l*. Здесь мы предполагаем, что записи изменений имеют форму (*i*,δ), где *i* – номер ячейки памяти и δ символ, который размещен в этой ячейке. Затем, мы сортируем пары из *W* по их левым элементам в соответствии с так называемыми *ключами сортировки* так, что, если два ключа сортировки равны, тогда соответствующие пары хранятся упорядоченными. Определяющим является то, что сортировка выполняется посредством использования эффективной сети сортировки такой, например, как сеть сортировки Батчера. Следует подчеркнуть, что всякий раз, когда две пары сравниваются и переставляются (или нет), тогда они заново шифруются посредством *Е* (так что противник не может «понять» были ли они переставлены или нет). Это гарантирует то, что вся процедура сортировки не дает никакой информация противнику. Как только сортировка завершена, алгоритм просматривает *W* для нахождения всех мест нахождения элементов с одним и тем же ключом сортировки и сохраняет последнее (которое является новым), в большом фиктивном значении (*l*+1,...). Теперь, мы посредством ключа сортировки заново сортируем пары из *W* и получаем последовательность, в которой первые *l* элементов содержат модифицированный документ *D'*. В заключение, мы устанавливаем *Е*1 для хранения в ней зашифрованного блока данных *D'* и исключаем первые *l* элементов последовательности *Е*2.

При использовании *AKS*-сети сортировки вышеприведенная реализация инкрементального алгоритма шифрования требует *O*(*l*log *l*) шагов (в то время как, если мы используем сеть Батчера, то получаем сумму из *О*(*l*)+(*l*lоg2*l*) шагов). Эти шаги могут быть распределены равномерно среди *l* операций модификации, обеспечивающих желаемую сложность.

Как было упомянуто выше, каждый из рассматриваемых алгоритмов можно адаптировать для реализации схемы инкрементального шифрования для операций вставки/удаления (единственного символа), которая является эффективной в строгом смысле. Для этого длина открытых данных устанавливается в некоторых предопределенных пределах (например, между *l*/2 и 2*l*). А именно, схема инкрементального шифрования состоит из трех последовательностей *Е*1, *Е*2 и *Е*3, где *Е*1 и *Е*2 – определены выше, а *Е*3 - шифрование рабочей области для некоторого забывающего моделирующего устройства. Также как и как выше, алгоритм работает в периодах, состоящих из *l* изменений каждый. В каждом периоде, инкрементальный алгоритм выполняет *l* изменений предыдущего периода. Это делается посредством моделирования *RAM*- программы, которая содержит структуру данных, обеспечивающую эффективное выполнение операций вставки/удаления, например, 2-3- дерево.

**Вопросы стойкости инкрементальных схем**

Обеспечение безопасности в традиционных схемах подписи и шифрования сводится к исследованию поведения противника, который, не зная секретного ключа, пытается осуществить злоумышленные действия в отношении этих схем. Например, в схемах подписи, в том числе, требуется, чтобы противник, не зная ключа, не мог бы подделывать подписи.

В случае инкрементальной криптографии нас будет интересовать информация о *предыдущих версиях документа*, которая может быть получена законным пользователем при проверке текущего документа вместе с текущей криптографической формой. То есть, предположим, что мы имеем блок данных *D* вместе с его обработанной криптографической формой и мы говорим, что *D* был получен из некоторого другого блока данных, именуемого *D'* посредством удаления единственного символа. *Абсолютная секретность* должна означать то, что противник ничего не может сказать о местоположении удаленного символа. *Частичная секретность* может означать, что противник не может ничего сообщить о значении удаленного символа, но может иметь некоторую информацию относительно его местоположения.

Абсолютная секретность представляет собой естественный интерес в контексте исследования стойкости схем подписи. Предположим, что мы используем инкрементальную схему подписи для многих пользователей. Желательно, чтобы ни один из этих пользователей не мог узнать что-либо из подписи другого пользователя. Частичная секретность также представляет интерес. Предположим, что абонент**А** имеет некоторую инкрементальную рабочую схему, с помощью которой он подписал некоторый документ. Ясно, что **А** не должен заботиться о том, узнает ли **В**, которому он дал такую инкрементальную рабочую схему, что **А** подписал этот документ, используя подпись к некоторому другому документу, т.е. **В** не может выяснить любые детали относительно предыдущего использования инкрементальной схемы.

Определение абсолютной и частичной секретности может быть легко дано с использованием следующей стандартной парадигмы. А именно, для данного блока данных *D* и подписи к нему нельзя различить, была ли подпись сделана как реакция на команду создания или как реакция на команду модификации. При определении частичной секретности главным аргументом является то, что единственной высвобождаемой информацией является количество изменений для данного блока *D*.

Схема инкрементальной аутентификации, рассматриваемая выше, удовлетворяет определению частичной секретности.

**Применение инкрементальных алгоритмов для защиты от вирусов**

Предлагаемые схемы аутентификации для защиты от вмешивающихся противников при антивирусной защите могут быть использованы следующим образом.

Пусть каждый защищаемый файл вместе с деревом аутентификационных признаков хранится в общедоступной памяти. При подходящем выборе сложностных параметров затраты памяти для хранения дерева аутентификационного признака (АП) могут быть незначительными по отношению к самому файлу. Например, мы можем разделить файл на блоки длины *s2*, где *s* - длина АП (и соответствующего ключа) базовой схемы аутентификации сообщений). Для *L*-битного файла, мы получим дерево АП с *L*/*s2* ветвями, которое может быть закодировано двоичной строкой размером *O*(*L*/*s*). Для каждого файла пользователю необходимо хранить только *O*(*s*) битов в локальной защищенной памяти. Эти биты используются для хранения ключа схемы аутентификации, имени файла и текущего счетчика версий.

Как только файл подвергся изменениям, дерево АП (хранящееся в небезопасной памяти) и счетчик версий (хранящийся в локальной защищенной памяти) изменяются в соответствии с вышеописанным инкрементальным алгоритмом аутентификации. В любое время, как только целостность файла подверглась сомнению, можно верифицировать корректность дерева АП очевидным образом.

Базовая схема аутентификации сообщений может быть любой из стандартных. Например, режим генерации имитовставки алгоритма ГОСТ 28147-89 или любые схемы аутентификации, использующие псевдослучайные функции.

Кроме того, предлагаемая схема имеет дополнительное свойство безопасности, а именно, даже, если противник может просматривать дерево АП и даже изменять его, схема аутентификации по-прежнему остается безопасной.

Другое направление инкрементальной криптографии заключается в использовании ее методов при защите программ, которая определяется в терминах предыдущего раздела. А именно, такая конструкция состоит из процессора с ограниченным количеством локальной памяти для хранения и доступа к информации, хранящейся в удаленной незащищенной памяти. Моделирование должно быть забывающим в том смысле, что фактическая модель доступа не дает никакой информации об оригинальной модели доступа. Перенос забывающего моделирования *RAM*-программ в инкрементальную схему шифрования очевидно. Роль процессора играет пользователь, в то время как роль удаленной памяти ассоциирована с шифрованием. Схема защиты программ с полилогарифмическими затратами существует и, используя эти результаты, можно показать, что эффективность схемы инкрементального шифрования при защите программ не хуже, чем ее эффективность при защите электронных данных

Однако идеи, используемые для защиты программ можно адаптировать для получения инкрементальной схемы шифрования для операций вставки/удаления (единственного символа), которые является эффективным в строгом смысле, т.е. как число шагов моделирования на одну оригинальную операцию. Адаптация достигается «конвейерной обработкой», описание которой дано выше.