

## СОКРЫТИЕ ДАННЫХ В КЛАСТЕРНЫХ ФАЙЛОВЫХ СИСТЕМАХ

### Введение

Современные методы стеганографической защиты позволяют создавать эффективные механизмы сокрытия не только смыслового содержания информационных данных, но и самого факта их существования, а также возможной передачи этих данных между уполномоченными абонентами.

Известные методы цифровой стеганографии используют т.н. контейнеры – цифровые данные с высоким уровнем естественной избыточности (цифровые фото- и видеоизображения, аудиозаписи, текстовые документы и пр.), которые предназначены для сокрытия в них передаваемых сообщений. Информационные данные встраиваются в контейнеры посредством специального преобразования (стеганографического кодирования) с использованием ключевых данных. На приемной стороне уполномоченный пользователь (владеющий секретным ключом) осуществляет извлечение информационных данных. Для неуполномоченного пользователя (противника, злоумышленника) передаваемый заполненный контейнер (стеганограмма) неотличим от пустых контейнеров, которые, как предполагается, в избытке передаются по открытым каналам связи<sup>1</sup>.

Среди последних достижений цифровой стеганографии следует выделить методы сокрытия информационных данных, основанные на использовании особенностей в построении различных файловых систем, установленного порядка организации, хранения и именования данных на физических носителях информации. Другими словами, эти методы не используют традиционные цифровые контейнеры, обладающие высоким уровнем *естественной избыточности*. Избыточность в данном случае является *искусственной*, поскольку ее появление обусловлено техническими особенностями применяемых способов представления, хранения, передачи и отображения цифровых данных. Чтобы подчеркнуть указанное отличие, будем относить рассматриваемые методы к так называемой *технической стеганографии*.

Цель работы – анализ методов сокрытия данных в кластерных файловых системах, исследование их возможностей по построению эффективных механизмов стеганографической защиты информации.

### Современные файловые системы, классификация и основные функции

Рассмотрим структуру наиболее известных и распространенных файловых систем (ФС) и принятые в них правила хранения цифровых данных (фалов данных) на различных физических носителях.

Под *файловой системой* (file system) понимают установленный порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерных системах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т.п. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов. Конкретная файловая система определяет размер имен файлов и (каталогов), максимальный возможный размер файла и раздела, набор атрибутов файла. Некоторые ФС

---

<sup>1</sup> Под каналом связи здесь и далее понимается любая среда распространения, хранения или отображения цифровых данных, например, это могут быть сетевые хранилища мультимедийных ресурсов, технологии электронной почты и предоставляемые ею услуги по пересылке и получению электронных сообщений, приложения по обмену фотографиями и видеозаписями (Instagram), социальные сети, технологии цифрового теле и радиовещания, в том числе по протоколам Интернета (IPTV), а также многое другое.

предоставляют сервисные возможности, например разграничение доступа или шифрование файлов.

Таким образом, ФС связывает носитель информации с одной стороны и API<sup>2</sup> для доступа к файлам – с другой. Когда прикладная программа обращается к файлу, она не имеет данных о том, каким образом расположена информация в конкретном файле, так же, как и на каком физическом типе носителя (CD, жестком диске, магнитной ленте, блоке флеш-памяти или другом) он записан. Все, что имеет программа – это имя файла, его размер и атрибуты. Эти данные она получает от драйвера файловой системы. Именно файловая система устанавливает, где и как будет записан файл на физическом носителе (например, жестком диске).

С точки зрения операционной системы (ОС), весь диск представляет собой набор кластеров (как правило, размером 512байт и больше). Драйверы ФС организуют кластеры в файлы и каталоги (реально являющиеся файлами, содержащими список файлов в этом каталоге). Эти же драйверы отслеживают, какие из кластеров в настоящее время используются, какие свободны, какие помечены как неисправные.

Однако ФС не обязательно напрямую связана с физическим носителем информации. Существуют виртуальные ФС, а также сетевые ФС, которые являются лишь способом доступа к файлам, находящимся на удаленном компьютере.

*Основные функции ФС* нацелены на решение следующих задач:

- именование файлов;
- программный интерфейс работы с файлами для приложений;
- отображения логической модели файловой системы на физическую организацию хранилища данных;
- организация устойчивости файловой системы к сбоям питания, ошибкам аппаратных и программных средств;
- содержание параметров файла, необходимых для правильного его взаимодействия с другими объектами системы (ядро, приложения и пр.).

В многопользовательских системах появляется еще одна задача: защита файлов одного пользователя от несанкционированного доступа другого пользователя, а также обеспечение совместной работы с файлами, к примеру при открытии файла одним из пользователей, для других этот же файл временно будет доступен в режиме «только чтение».

ФС в самой общей классификации могут быть разделены на дисковые ФС, распределенные и специальные ФС (см. рис. 1). Наибольшее развитие в вычислительной технике традиционно получили дисковые накопители, структура которых в общем виде представлена на рис. 2. Данные на дисковых накопителях записываются по т.н. дорожкам.

Совокупность дорожек условно разбивается на т.н. геометрические сектора, при этом часть дорожки конкретного геометрического сектора называется сектором дорожки. Основной логической единицей хранения данных в таблице размещения файлов дисковых файловых систем является кластер.

*Кластер* (англ. cluster) – логическая единица хранения данных в таблице размещения файлов, объединяющая группу секторов дорожки. Например, на дисках с размером секторов в 256 байт, 256-байтный кластер содержит один сектор, тогда как двухкилобайтный кластер содержит восемь секторов. Как правило, кластер – это наименьшее место на диске, которое может быть выделено для хранения файла.

Понятие кластер используется в файловых системах FAT, NTFS, а также HFS Plus. Другие файловые системы оперируют схожими понятиями (зоны в Minix, блоки в Unix и пр.). В некоторых файловых системах Linux (ReiserFS, Reiser4, Btrfs), BSD (FreeBSD UFS2) последний блок файла может быть поделен на подфрагменты, в которые могут быть

---

<sup>2</sup> API (application programming interface) – интерфейс программирования приложений (иногда интерфейс прикладного программирования)

помещены «хвосты» других файлов. В NTFS маленькие файлы могут быть записаны в Master File Table (MFT). В файловой же системе FAT из-за примитивного алгоритма степень фрагментации постоянно растет и требуется периодическая дефрагментация. Маленький кластер лучше подходит для маленьких файлов. Так экономнее расходуется место. Большой кластер позволяет достичь более высоких скоростей, но на мелких файлах место будет использоваться нерационально (многие сектора будут не полностью заполненными, но будут считаться занятыми).

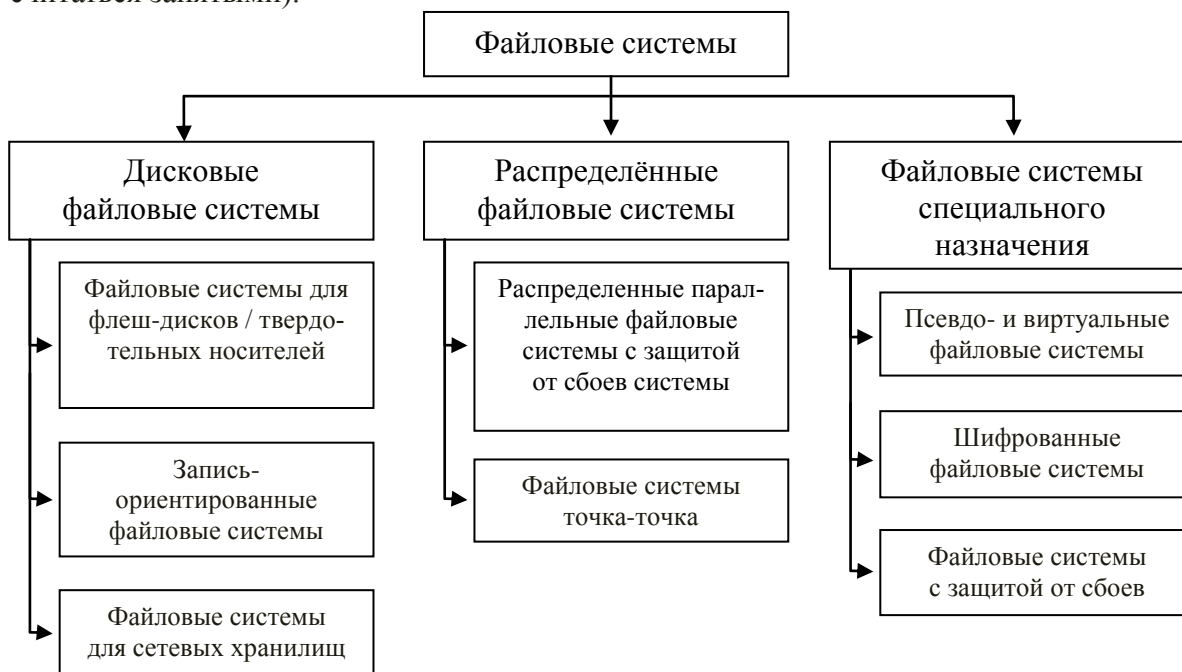


Рис. 1. Общая классификация файловых систем

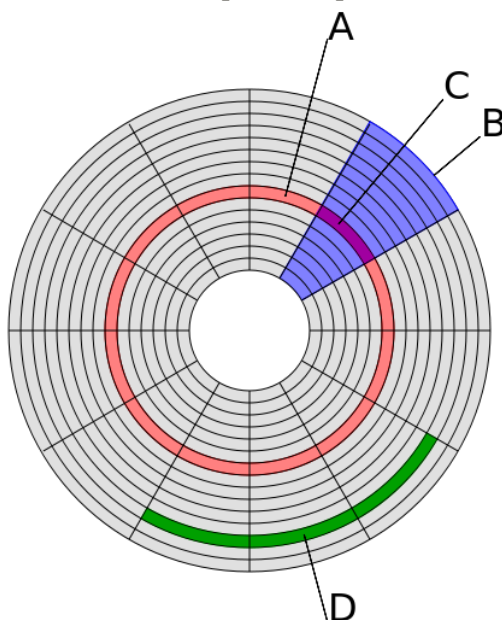


Рис. 2. Структура диска: А – дорожка; В – геометрический сектор;  
С – сектор дорожки; D – кластер

Дисковые файловые системы обычно являются поток-ориентированными. Файлы в поток-ориентированных файловых системах представляются последовательностью битов, часто предоставляют такие функции, как чтение, запись, изменение данных и произвольный

доступ. Наиболее распространенными поток-ориентированными файловыми системами являются FAT (File Allocation Table) – три ее различных типа (FAT16, FAT32, FAT64) и NTFS (от англ. New Technology File System – «файловая система новой технологии»). Рассмотрим их подробнее.

#### *Файловая система FAT.*

Файловая система FAT (File Allocation Table) разработана Биллом Гейтсом и Марком Макдональдом в 1977 году. Существует четыре версии FAT – FAT8, FAT12, FAT16 и FAT32:

- FAT12 – поддерживает очень небольшие объемы дисков, поэтому сейчас она применяется только на дискетах.

- FAT16 – используется на винчестерах и поддерживает диски объемом до 2 Гб, поэтому сейчас данная файловая система практически не используется.

- FAT32 – теоретически поддерживаются диски объемом до 2 Тб. Поддерживается начиная с операционной системы Windows 95 OSR2. Данная файловая система достаточно популярна, хотя в последние годы многие пользователи предпочитают использовать NTFS (New Technology File System).

Таким образом, различные версии FAT отличаются разрядностью записей в дисковой структуре, то есть количеством бит, отведенных для хранения номера кластера. На основе FAT разработана новая файловая система exFAT (extended FAT), используемая преимущественно для флеш-накопителей.

Структура файловой системы FAT представлена на рис. 3. Она состоит:

- *загрузочный сектор*, который располагается в начале раздела диска с файловой системой FAT и необходим для начальной загрузки компьютера. Также в нем располагается информация о параметрах данного раздела;

- *таблица размещения файлов (File Allocation Table)*. Вся область данных диска разделена на кластеры – блоки, размер которых задается при форматировании диска. На дискете, например, размер кластера равен 512 байт. А на винчестерах с объемом диска более 32 Гб размер кластера равен 32 Кб. Каждый файл и каталог занимает один или несколько кластеров. Таким образом, образуются цепочки кластеров. В таблице размещения файлов каждый кластер помечается специальным образом. Размер метки в битах для каждого кластера указывается в названии файловой системы. Т.е. для файловой системы FAT16 размер метки равен 16 байт, для FAT32 – 32 и т.д. Всего существует три типа меток для кластеров: свободный кластер – кластер, в который будут записываться новые файлы и каталоги; занятый кластер – в метке указывается следующий кластер в цепочке (если цепочка кластеров заканчивается, то кластер помечается особой меткой); BAD-блок – кластер с ошибками доступа (помечается при форматировании диска, чтобы в последующем исключить к нему доступ). Повреждение таблицы размещения файлов полностью уничтожает структуру файловой системы, поэтому на диске всегда хранится две копии таблицы;

- *файловые записи*. Непосредственно после окончания последней таблицы FAT следует область данных, содержащая файлы и папки. Каталог FAT (папка, директория) является обычным файлом, помеченным специальным атрибутом. Данными (содержимым) такого файла в любой версии FAT является цепочка 32-байтных файловых записей (записей каталога). Каталог не может штатно содержать два файла с одинаковым именем. Если программа проверки диска обнаруживает искусственно созданную пару файлов с идентичным именем в одном каталоге, один из них переименовывается;

- *корневой каталог*. Область диска, в котором располагается информация о корневом каталоге. Размер ее ограничен, поэтому в корневом каталоге диска может находиться не более 512 файлов и подкаталогов.

Загрузочный сектор	Таблица FAT (1-я копия)	Таблица FAT (2-я копия)	Корневой каталог	Область данных
--------------------	-------------------------	-------------------------	------------------	----------------

Рис. 3. Структура файловой системы FAT

Основным преимуществом ФС FAT является ее простота и совместимость со старыми операционными системами. Для этой ФС существует большое количество подробной документации. Сбои в системе часто приводят к повреждению одного или нескольких файлов. Однако при серьезных повреждениях, восстановить информацию гораздо проще, чем в случае с NTFS.

#### *Файловая система NTFS.*

NTFS (от англ. New Technology File System – «файловая система новой технологии») – стандартная файловая система для семейства операционных систем Microsoft Windows NT. NTFS заменила использовавшуюся в MS-DOS и Microsoft Windows файловую систему FAT. NTFS поддерживает систему метаданных и использует специализированные структуры данных для хранения информации о файлах для улучшения производительности, надежности и эффективности использования дискового пространства. NTFS хранит информацию о файлах в главной файловой таблице – Master File Table (MFT). NTFS имеет встроенные возможности разграничения доступа к данным для различных пользователей и групп пользователей (списки контроля доступа – Access Control Lists (ACL)), а также назначать квоты (ограничения на максимальный объем дискового пространства, занимаемый теми или иными пользователями). NTFS использует систему журналирования USN для повышения надежности файловой системы.

Спецификации файловой системы закрыты. Это создает определенные трудности при реализации ее поддержки в сторонних продуктах, не принадлежащих Microsoft, в частности разработчикам драйверов для свободных операционных систем приходится заниматься обратной разработкой системы.

*Структура раздела.* Как и любая другая система, NTFS делит все полезное место на кластеры – блоки данных, используемые одновременно. NTFS поддерживает почти любые размеры кластеров – от 512 байт до 64 Кбайт, неким стандартом же считается кластер размером 4 Кбайт. Диск NTFS условно делится на две части. Первые 12 % диска отводятся под так называемую MFT зону – пространство, в которое растет метафайл MFT (об этом ниже). Запись каких-либо данных в эту область невозможна. MFT-зона всегда держится пустой – это делается для того, чтобы самый главный, служебный файл (MFT) не фрагментировался при своем росте. Остальные 88 % диска представляют собой обычное пространство для хранения файлов. Свободное место диска включает в себя все физически свободное место – незаполненные куски MFT-зоны туда тоже включаются. Когда файлы уже нельзя записывать в обычное пространство, MFT-зона сокращается, освобождая таким образом место для записи файлов. При освобождении места в обычной области MFT зона может снова расшириться.

*MFT и его структура.* Каждый элемент ФС представляет собой файл – даже служебная информация. Самый главный файл на NTFS называется MFT, или Master File Table – общая таблица файлов. Именно он размещается в MFT зоне и представляет собой централизованный каталог всех остальных файлов диска, и, как не парадоксально, себя самого. MFT поделен на записи фиксированного размера (обычно 1 Кбайт), и каждая запись соответствует какому либо файлу (в общем смысле этого слова). Первые 16 файлов носят служебный характер и недоступны операционной системе – они называются метафайлами, причем самый первый метафайл – сам MFT. Эти первые 16 элементов MFT – единственная часть диска, имеющая фиксированное положение. Интересно, что вторая копия первых трех записей, для надежности – они очень важны – хранится ровно посередине диска. Остальной MFT-файл может располагаться, как и любой другой файл, в произвольных местах диска – восстановить его положение можно с помощью его самого, "зацепившись" за самую основу – за первый элемент MFT.



**Метафайлы.** Первые 16 файлов NTFS (метафайлы) носят служебный характер. Каждый из них отвечает за какой-либо аспект работы системы. Преимущество настолько модульного подхода заключается в поразительной гибкости – например, на FAT-е физическое повреждение в самой области FAT фатально для функционирования всего диска, а NTFS может сместить, даже фрагментировать по диску, все свои служебные области, обойдя любые неисправности поверхности – кроме первых 16 элементов MFT.

**Файлы и потоки.** Прежде всего, обязательный элемент – запись в MFT, ведь, как было сказано ранее, все файлы диска упоминаются в MFT. В этом месте хранится вся информация о файле, за исключением собственно данных. Имя файла, размер, положение на диске отдельных фрагментов и т.д. Если для информации не хватает одной записи MFT, то используются несколько, причем, не обязательно подряд. Потоки данных NTFS (известные также как альтернативные потоки данных) и доступ к списку ресурсов на базе разрешений Access-based Enumeration (ABE). Альтернативные потоки данных дают возможность добавлять к файлу скрытую информацию, такую как сведения о файле. Скорее всего, вам не придется задействовать скрытые потоки данных, однако злоумышленники могут использовать эту технологию против вас, так что следует иметь представление о ней и о том, как она может работать.

**Фрагментация и дефрагментация в NTFS.** NTFS – система, которая предрасположена к фрагментации. Однако все внутренние структуры построены таким образом, чтобы фрагментация не препятствовала быстрому поиску фрагментов данных. Диск NTFS поделен на две зоны. В начале диска идет MFT зона – зона, где располагается MFT, которая занимает минимум 12 % диска, и запись данных в эту зону невозможна.

Таким образом, наиболее распространенными ФС на сегодняшний день являются FAT и NTFS. Анализируя достоинства и недостатки этих двух ФС, следует отметить следующее:

- NTFS является более надежной ФС, устойчивой к различным сбоям и незначительным ошибкам. Однако требования к объему оперативной памяти по сравнению с FAT 32 этой ФС выше, кроме того, обработка файлов осуществляется с меньшей скоростью;

- FAT 32 является простой и очень быстрой ФС, которая обеспечивает меньший износ физических носителей. Однако надежность этой ФС ниже, чем NTFS, кроме того, существуют ограничения по объему файлов, которые могут храниться в такой ФС;

- обе ФС хранят данные в кластерах, минимальный размер которых равен 512 байт. Как правило, обычный размер кластера равен 4 Кбайт. При этом скорость работы NTFS резко снижается при заполнении диска на 80 – 90 %. Это связано с фрагментацией служебных и рабочих файлов. В FAT 32 фрагментация рабочей области диска происходит и на более ранних этапах. Как и в NTFS, фрагментация сильно снижает производительность.

Рассмотрим методы стеганографической защиты, основанные на использовании кластерных ФС.

### **Методы сокрытия данных в кластерных файловых системах**

Первые методы встраивания данных в кластерные файловые системы были предложены в статьях Hassan Khan [1 – 2]. В этих работах для встраивания информации предложено использовать искусственную избыточность, которая возникает в результате специального способа кодирования, представления и именования файлов данных в кластерных файловых системах. Дальнейшее развитие эти идеи получили в работе [3], где специалистами Каунасского технологического университета (Литва) улучшены отдельные показатели стеганозащиты и предложены оригинальные алгоритмы встраивания и извлечения информации.

Основная идея предложенного в [3] метода встраивания данных состоит в использовании нескольких покрывающих файлов (cover file) и сокрытии секретного сообщения при помощи кодирования относительных позиций кластеров этих файлов друг относительно друга. В таком случае игнорируются все другие файлы в файловой системе и анализируются только кластеры, принадлежащие всем покрывающим файлам.

Если создать массив индексов кластеров в файловой системе и отсортировать его в порядке возрастания, можем получить последовательность кластеров, принадлежащих только покрывающим файлам. Тогда, например, если для некоторых индексов массива за кластером одного покрывающего файла следует кластер другого покрывающего файла, можем присвоить логический бит "1" скрытого сообщения, и наоборот. Другими словами, стеганокодирование состоит в изменении порядка следования кластеров покрывающих файлов в зависимости от значения встраиваемых информационных бит. Кроме того, в случае использования более чем двух покрывающих файлов, можем назначить относительные значения для каждого cover file и для каждого индекса массива отсортированных кластеров вычислить разницу между номерами предыдущих и текущих файлов. В этом случае можно встроить больше, чем один бит секретного сообщения в один кластер файловой системы, т.е. пропускная способность стеганографического канала передачи данных с увеличением числа покрывающих файлов возрастает. Соответственно, имена покрывающих файлов должны храниться в секрете, т.е. являются элементом секретного ключа, и принимающая уполномоченная сторона должна знать имена и точный порядок этих файлов для того, чтобы успешно извлечь скрытое сообщение.

Рассмотрим известные алгоритмы сокрытия и извлечения данных с помощью двух и более покрывающих файлов [3], проанализируем возможности построения с их помощью эффективных средств стеганографической защиты.

#### Алгоритм встраивания данных

**Шаг 1.** Информационное сообщение представляется в виде битового массива:

$$M = \{b_0, b_1, \dots, b_{n-1}\}, b_i \in \{0, 1\}. \quad (1)$$

**Шаг 2.** На кластерном носителе выбираются  $p = 2^m$ ,  $m \in N$  покрывающих файлов (cover files), обозначим их как:

$$F_0, F_1, \dots, F_{p-1}. \quad (2)$$

Порядок размещения кластеров покрывающих файлов будет скрывать информационное сообщение, т.е. после встраивания информации эти файлы нельзя стирать, перемещать или модифицировать.

Натуральное число  $m$  и имена покрывающих файлов являются частью секретного ключа. Важен также порядок упорядочивания cover files.

**Шаг 3.** Формируется массив  $C$  номеров кластеров покрывающих файлов:

$$C = \begin{pmatrix} c_{0,0} & c_{0,1} & \dots & c_{0,L_0-1} & & & \\ c_{1,0} & c_{1,1} & \dots & \dots & \dots & \dots & c_{1,L_1-1} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ c_{p-1,0} & c_{p-1,1} & \dots & \dots & c_{p-1,L_{p-1}-1} & & \end{pmatrix}, \quad (3)$$

где каждая строка массива содержит номера кластеров соответствующего файла. Например, файлу  $F_i$  соответствует  $i$ -я строка массива  $C$ , т.е. номера кластеров  $i$ -го покрывающего файла могут быть представлены в виде массива-строки:

$$C_{F_i} = \{c_{i,0}, c_{i,1}, \dots, c_{i,L_i-1}\}, \quad (4)$$

где  $L_i$  – число кластеров в  $i$ -м покрывающем файле  $F_i$ .

Если при встраивании информации необходимо сохранить без изменения содержимое покрывающих файлов, тогда требуется, чтобы выполнялись условия:

$$\forall i: L_i \geq k, k = n/m. \quad (5)$$

**Шаг 4.** Формируется массив  $D$  номеров пустых кластеров файловой системы:

$$D = \{c_1, c_2, \dots, c_{L_D}\}, \quad (6)$$

причем

$$c_1 < c_2 < \dots < c_{L_D}. \quad (7)$$

Число  $L_D$  равно количеству пустых кластеров файловой системы, причем требуется, чтобы выполнялось неравенство

$$L_D \geq \sum_{i=0}^{p-1} L_i. \quad (8)$$

**Шаг 5.** Информационное сообщение разбивается на блоки по  $m$  бит каждый:

$$M = \{B_1, B_2, \dots, B_k\}, \quad (9)$$

где  $k = \lceil n/m \rceil$  и если  $k = n/m$ , тогда

$$B_1 = \{b_0, b_1, \dots, b_{m-1}\}, B_2 = \{b_m, b_{m+1}, \dots, b_{2m-1}\}, \dots, B_k = \{b_{(k-1)m}, b_{(k-1)(m+1)}, \dots, b_{km-1}\}. \quad (10)$$

Если  $k < n/m$ , тогда последний блок дополняется нулями:

$$B_1 = \{b_0, b_1, \dots, b_{m-1}\}, B_2 = \{b_m, b_{m+1}, \dots, b_{2m-1}\}, \dots, B_k = \{b_{(k-1)m}, b_{(k-1)(m+1)}, \dots, b_{n-1}, \underbrace{0, 0, \dots, 0}_{km-n}\}. \quad (11)$$

Каждый блок  $B_i$ ,  $i = 1, 2, \dots, k$  интерпретируется как натуральное число, где битовый блок является двоичным представлением этого числа. Очевидно, что в такой интерпретации

$$\forall i: 0 \leq B_i \leq p-1. \quad (12)$$

**Шаг 6.** Задается натуральное число  $B_0$ , которое фактически выступает вектором инициализации и может быть частью секретного ключа, либо последним блоком предварительно встроенного сообщения.

**Шаг 7.** Устанавливаем  $i = 0$ , и для всех  $i = 0, 1, \dots, p-1$  устанавливаем  $j_i = 0$ .

**Шаг 8.**

$$i = i + 1;$$

$$N_i = B_{i-1} + B_i \bmod p, \quad 0 \leq N_i \leq p-1;$$

$c_i = c_{N_i j_{N_i}}$ , т.е. перезаписываем  $j_{N_i}$ -й кластер покрывающего файла  $F_{N_i}$  на место пустого кластера с номером  $c_i$  (из массива  $D = \{c_1, c_2, \dots, c_{L_D}\}$ );

$$j_{N_i} = j_{N_i} + 1.$$

**Шаг 9.** Повторяем шаг 8 еще  $k-1$  раз.

**Шаг 10.** Записываем оставшиеся кластеры покрывающих файлов  $F_0, F_1, \dots, F_{p-1}$  в произвольном порядке на свободные места файловой системы (при необходимости).

В результате выполнения алгоритма первые  $k$  свободных кластеров файловой системы будут записаны кластерами покрывающих файлов. Причем на  $i$ -й пустой кластер записывается  $j_{N_i}$ -й кластер файла  $F_{N_i}$ , где номер файла вычисляется рекурсивно  $N_i = B_{i-1} + B_i \bmod p$ , с начальным значением  $B_0$  (вектор инициализации).

### Алгоритм извлечения данных

**Шаг 1.** Вводится секретный ключ: вектор инициализации  $B_0$ , названия и порядок упорядочивания покрывающих файлов  $F_0, F_1, \dots, F_{p-1}$ , длина встроенного сообщения  $n$ .

**Шаг 2.** Формируется массив  $D$  номеров кластеров покрывающих файлов:

$$D = \{c_1, c_2, \dots, c_{L_D}\}, \quad (13)$$

причем



$$c_1 < c_2 < \dots < c_{L_D}. \quad (14)$$

Каждый номер кластера из массива  $D$  соотносится только с одним кластером одного покрывающего файла. Это соответствие определяется логикой встраивания информации и используется на следующих шагах для извлечения данных.

**Шаг 3.** Устанавливаем  $i = 0$  и для всех  $i = 0, 1, \dots, p-1$  устанавливаем  $j_i = 0$ .

**Шаг 4.**

$$i = i + 1;$$

Из файловой системы считываем кластер с индексом  $c_i$ ;

Значение  $N_i$  устанавливаем равным номеру того файла, чьи данные хранятся в кластере с индексом  $c_i$ , т.е.  $c_i = c_{N_i j_{N_i}}$ ;

$$B_i = N_i - B_{i-1} \bmod p, \quad 0 \leq B_i \leq p-1;$$

$$j_{N_i} = j_{N_i} + 1.$$

**Шаг 5.** Повторяем шаг 4 еще  $k-1$  раз.

**Шаг 6.** Каждое натуральное число  $B_i$ ,  $i = 1, 2, \dots, k$  интерпретируется как битовый блок длины  $m$  бит каждый:

$$B_1 = \{b_0, b_1, \dots, b_{m-1}\}, B_2 = \{b_m, b_{m+1}, \dots, b_{2m-1}\}, \dots, B_k = \{b_{(k-1)m}, b_{(k-1)(m+1)}, \dots, b_{km-1}\}. \quad (15)$$

**Шаг 7.** Из битовых блоков формируется информационное сообщение

$$M = \{b_0, b_1, \dots, b_{n-1}\}, \quad b_i \in \{0, 1\}. \quad (16)$$

Т.е. если  $k < n/m$ , тогда последний блок «обрезается» – его последние  $km - n$  бит не используются.

Рассмотрим пример использования рассмотренных алгоритмов встраивания и извлечения информационных данных.

### Пример встраивания данных

**Шаг 1.** Пусть информационное сообщение представлено в виде битового массива:

$$M = \{1, 0, 0, 1, 1, 1, 0, 0\}, \quad n = 8.$$

**Шаг 2.** На кластерном носителе выберем  $p = 2^2 = 4$  покрывающих файла  $F_0, F_1, F_2, F_3$  ( $m = 2$ ). Пусть, например, это будут файлы с именами: «A.txt», «B.txt», «C.txt», «D.txt». Именно в таком порядке покрывающие файлы и будем использовать.

**Шаг 3.** Формируем массив  $C$  номеров кластеров покрывающих файлов. Формально запишем его, например, так (конкретный вид задается конкретным состоянием файлового носителя информации):

$$C = \begin{pmatrix} 101 & 105 & 117 & 168 & 189 \\ 115 & 116 & 118 & 188 \\ 110 & 111 & 112 & 113 \\ 191 & 192 & 194 & 195 & 196 & 197 \end{pmatrix},$$

т.е. данные файла  $F_0$  с именем «A.txt» хранятся в кластерах с номерами  $C_{F_0} = \{101, 105, 117, 168, 189, 191, 225\}$ , данные файла  $F_1$  с именем «B.txt» хранятся в кластерах с номерами  $C_{F_1} = \{115, 116, 118, 188\}$ , и т.д. Файл «A.txt» занимает 7 кластеров, файл «B.txt» – 4 кластера и т.д.

Очевидно, что условие  $\forall i: L_i \geq k = \lceil n/m \rceil = 4$  выполняется.

**Шаг 4.** Формируем массив  $D$  номеров пустых кластеров файловой системы.

Пусть, например, свободными являются кластеры с номерами:

$D = \{102, 103, 106, 109, 114, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, \dots\}$ ,

Причем  $102 < 103 < 106 < 109 < 114 < \dots$ . Кроме того, положим, что число свободных кластеров больше, чем число кластеров, отведенных для хранения покрывающих файлов, т.е. выполняется условие:

$$L_D \geq \sum_{i=0}^{p-1} L_i. \quad (17)$$

**Шаг 5.** Информационное сообщение разбиваем на блоки по  $m = 2$  бит каждый:

$$B_1 = \{1, 0\}, B_2 = \{0, 1\}, B_3 = \{1, 1\}, B_4 = \{0, 0\}.$$

Каждый блок  $B_i$  интерпретируем как натуральное число, т.е. положим:

$$B_1 = 2, B_2 = 1, B_3 = 3, B_4 = 0.$$

**Шаг 6.** Зададим натуральное число  $B_0 = 2$ , которое будет выступать в качестве вектора инициализации и может быть частью секретного ключа.

**Шаг 7.** Устанавливаем  $i = 0$  и для всех  $i = 0, 1, \dots, p-1$  устанавливаем  $j_i = 0$ .

**Шаг 8.**

$$i = i + 1 = 1;$$

$$N_1 = B_0 + B_1 \bmod p = 2 + 2 \bmod 4 = 0;$$

$c_1 = c_{00} = 101$ , т.е. перезаписываем  $j_{N_0} = 0$ -й кластер покрывающего файла  $F_0$  на место пустого кластера с номером  $c_1 = 102$  (из массива  $D$ );

$$j_{N_0} = j_{N_0} + 1 = 1.$$

**Шаг 9.** Повторяем шаг 8 еще  $k-1 = 3$  раза. Получим:

$$i = i + 1 = 2;$$

$$N_2 = B_1 + B_2 \bmod p = 3 \bmod 4 = 3;$$

$c_2 = c_{30} = 191$ ,  $j_{N_3} = 0$ -й кластер покрывающего файла  $F_3$  перезаписываем на место пустого кластера с номером  $c_1 = 103$  (из массива  $D$ );

$$j_{N_3} = j_{N_3} + 1 = 1;$$

$$i = i + 1 = 3;$$

$$N_3 = B_2 + B_3 \bmod p = 4 \bmod 4 = 0;$$

$c_2 = c_{01} = 105$ ,  $j_{N_0} = 1$ -й кластер покрывающего файла  $F_0$  перезаписываем на место пустого кластера с номером  $c_2 = 106$  (из массива  $D$ );

$$j_{N_0} = j_{N_0} + 1 = 2;$$

$$i = i + 1 = 4;$$

$$N_4 = B_3 + B_4 \bmod p = 3 \bmod 4 = 3;$$

$c_3 = c_{31} = 192$ ,  $j_{N_3} = 1$ -й кластер покрывающего файла  $F_3$  перезаписываем на место пустого кластера с номером  $c_2 = 109$  (из массива  $D$ );

$$j_{N_3} = j_{N_3} + 1 = 2.$$

**Шаг 10.** Записываем оставшиеся кластеры покрывающих файлов  $F_0, F_1, F_2, F_3$  в произвольном порядке на свободные места файловой системы (при необходимости).

В результате выполнения алгоритма первые  $k = 4$  свободных кластеров файловой системы будут записаны кластерами покрывающих файлов. Массив  $C$  номеров кластеров покрывающих файлов примет вид:

$$C = \begin{pmatrix} 102 & 106 & 114 & 119 & 120 \\ 121 & 122 & 123 & 124 \\ 125 & 126 & 127 & 128 \\ 103 & 109 & 129 & 130 & 131 & 132 \end{pmatrix}.$$

Формально этот процесс представим в виде структуры файловой таблицы на рис. 4.

Cluster #	Content		Cluster #	Content
...	...		...	...
101	xxx.txt		101	xxx.txt
102	[free]		102	<b>A.txt</b>
103	[free]		103	<b>D.txt</b>
104	xxx.txt		104	xxx.txt
105	yyy.log		105	yyy.log
106	[free]		106	<b>A.txt</b>
107	xxx.txt		107	xxx.txt
108	<b>zzz.doc</b>		108	<b>zzz.doc</b>
109	[free]		109	<b>D.txt</b>
...	...		...	...

Рис. 4. Структура файловой таблицы до и после встраивания информации

#### Пример извлечения данных

**Шаг 1.** Вводим секретный ключ: вектор инициализации  $B_0 = 2$ , названия и порядок упорядочивания покрывающих файлов  $F_0, F_1, F_2, F_3$  ( $m = 2$ ). В нашем случае это будут файлы с именами: «A.txt», «B.txt», «C.txt», «D.txt», длина встроенного сообщения  $n$ .

**Шаг 2.** Формируется массив  $D$  номеров кластеров покрывающих файлов:

$$D = \{c_1, c_2, \dots, c_{L_D}\}, \quad (18)$$

причем

$$c_1 < c_2 < \dots < c_{L_D}. \quad (19)$$

Каждый номер кластера из массива  $D$  соотносится только с одним кластером одного покрывающего файла. Это соответствие определяется логикой встраивания информации и используется на следующих шагах для извлечения данных.

**Шаг 3.** Устанавливаем  $i = 0$  и для всех  $i = 0, 1, \dots, p-1$  устанавливаем  $j_i = 0$ .

**Шаг 4.**

$$i = i + 1 = 1;$$

Из файловой системы считываем первый кластер, в котором хранятся данные покрывающих файлов. Это кластер с индексом  $c_1 = 102$ ;

Значение  $N_1$  устанавливаем равным номеру того файла, чьи данные хранятся в кластере с индексом  $c_1 = 102$ , это файл  $F_0$ , следовательно  $c_1 = c_{00}$  и  $N_1 = 0$ ;

$$B_1 = N_1 - B_0 \bmod p = -2 \bmod 4 = 2;$$

$$j_{N_1} = j_{N_1} + 1 = 1.$$

**Шаг 5.** Повторяем шаг 4 еще  $k-1=3$  раза.

$$i = i + 1 = 2;$$

Из файловой системы считываем второй кластер, в котором хранятся данные покрывающих файлов. Это кластер с индексом  $c_2 = 103$ ;

Значение  $N_2$  устанавливаем равным номеру того файла, чьи данные хранятся в этом кластере, это файл  $F_3$ , следовательно  $c_2 = c_{30}$  и  $N_2 = 3$ ;

$$B_2 = N_2 - B_1 \bmod p = 1 \bmod 4 = 1;$$

$$j_{N_3} = j_{N_3} + 1 = 1.$$

$$i = i + 1 = 3;$$

Из файловой системы считываем третий кластер, в котором хранятся данные покрывающих файлов. Это кластер с индексом  $c_3 = 106$ ;

Следовательно  $c_3 = c_{01}$  и  $N_3 = 0$ ;

$$B_3 = N_3 - B_2 \bmod p = -1 \bmod 4 = 3;$$

$$j_{N_1} = j_{N_1} + 1 = 2.$$

$$i = i + 1 = 4;$$

Из файловой системы считываем четвертый кластер, в котором хранятся данные покрывающих файлов. Это кластер с индексом  $c_4 = 109$ ;

Следовательно  $c_4 = c_{31}$  и  $N_4 = 3$ ;

$$B_4 = N_4 - B_3 \bmod p = 0 \bmod 4 = 0;$$

$$j_{N_3} = j_{N_3} + 1 = 2.$$

**Шаг 6.** Каждое натуральное число  $B_i$ ,  $i = 1, 2, \dots, 4$  интерпретируется как битовый блок длины  $m = 2$  бит каждый:

$$B_1 = 2 = \{1, 0\}, B_2 = 1 = \{0, 1\}, B_3 = 3 = \{1, 1\}, B_4 = 0 = \{0, 0\}.$$

**Шаг 7.** Из битовых блоков формируется информационное сообщение

$$M = \{1, 0, 0, 1, 1, 0, 0\}.$$

### Оценка эффективности метода

Предложенный в [3] метод сокрытия данных в кластерных файловых системах с использованием нескольких покрывающих файлов очень прост в реализации. В отличие от других аналогичных методов процесс встраивания не создает никаких коллизий во время стеганокодирования, никакие другие файлы на физическом носителе не изменяются, что увеличивает скорость работы стеганосистемы.

Эксперименты авторов оригинальной статьи показывают [3], что объем встроенных данных составляет один бит на кластер при использовании двух покрывающих файлов. При использовании большего количества файлов объем встроенных данных определяется как  $\log_2 p = m$  бит на кластер. Наглядно это продемонстрировано на рис. 5, на котором приведены результаты сравнительного анализа пропускной способности различных стеганоалгоритмов с использованием кластерных ФС.

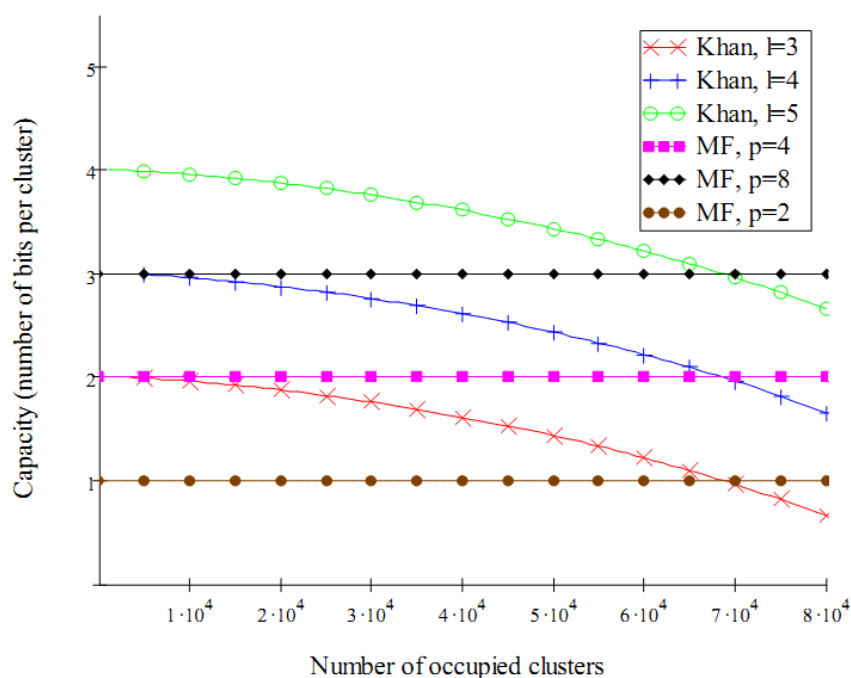


Рис. 5. Зависимость числа встраиваемых информационных бит на один кластер покрывающих файлов от числа задействованных кластеров (для различных стеганоалгоритмов)

На рис. 5 представлены результаты для методов, рассмотренных в статьях Hassan Khan [1 – 2], а также для метода из работы [3]. Параметр  $l$  устанавливает величину разрыва в кластерах при встраивании информационного бита. Зависимости, отмеченные «MF», соответствуют случаю использования метода из работы [3] с мультифайловым способом встраивания, т.е. рассмотрены варианты с  $p = 2^m$  покрывающими файлами,  $m = 1, 2, 3$ . Как видно из приведенных зависимостей, объем встраиваемой информации на один используемый кластер для мультифайлового метода не уменьшается с ростом числа использованных кластеров.

Рассмотренный метод не вносит дополнительной избыточности в процессе стеганокодирования, т.е. существующие методы анализа дискового пространства не смогут обнаружить факт встраивания информационных данных. Тем не менее, следует отметить, что реализация рассмотренных стеганоалгоритмов приводит к повышению фрагментации покрывающих файлов. В то же время высокая фрагментация является типичным свойством современных файловых систем. Следует также отметить, что модификация покрывающих файлов является критичной с точки зрения удаления встроенного информационного сообщения. Переименование, перемещение, копирование или удаление одного из cover files практически гарантированно уничтожит скрытую информацию.

Проведенный анализ показывает, что предложенный в [3] метод стеганозащиты не только фрагментирует покрывающие файлы, но также вводит специальный тип фрагментации – *переплетенные файлы*, когда кластеры одного покрывающего файла перемежаются кластерами другого покрывающего файла. Этот признак может быть положен в основу перспективных средств стеганографического анализа рассмотренных алгоритмов. Результаты исследования переплетения файлов приведены в работе [3], приведем некоторые из них. В частности, на рис. 6 представлены зависимости относительного числа переплетенных файлов. Например, первый столбец означает, что 4,33 % всех файлов переплетены, по крайней мере, с одним другим файлом. Кривая на рисунке представляет среднее число переплетений для каждого переплетенного файла в соответствующем объединении. Например, все файлы, которые переплетаются с, как минимум, четырьмя другими файлами (второй столбец), содержат в среднем 1460 переплетений.



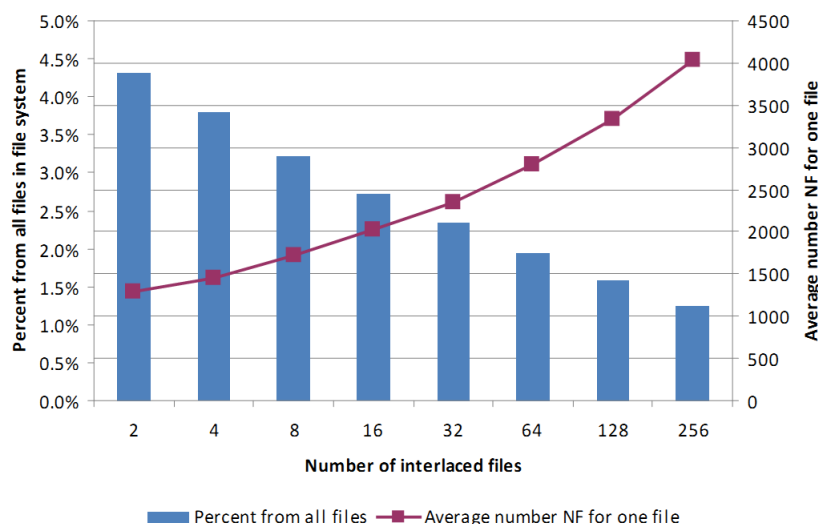


Рис. 6. Доля переплетенных файлов (диаграмма) и среднее число переплетений (кривая)

Приведенные зависимости характеризуют высокий уровень переплетения файлов в современных ФС. Следовательно, применение рассмотренного метода встраивания информации в кластерные ФС не окажет существенного изменения в распределении переплетенных файлов. Тем не менее, это утверждение нуждается в дополнительном обосновании и всесторонней проверке по результатам экспериментальных исследований.

## Выводы

1. Современные ФС устанавливают специальный порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерных системах, т.е. определяют формат содержимого и способ физического хранения файлов данных. Основной логической единицей хранения данных в таблице размещения файлов, объединяющей группу секторов дорожки, является кластер – это наименьшее место на диске, которое может быть выделено для хранения файла. Отдельные кластеры различных файлов могут быть перемешаны, т.е. размещаться в различном порядке следования. Эта вариативность в возможном размещении отдельных кластеров и составляет ту искусственно созданную в ФС избыточность, которую предлагается использовать в ряде методов стеганографической защиты.

2. Рассмотренные алгоритмы встраивания и извлечения информационных данных в кластерные ФС не вносят дополнительной избыточности в процессе стеганокодирования. Они основаны на изменении порядка следования отдельных кластеров специально выбранных покрывающих файлов. При передаче информационного сообщения покрывающие файлы не должны модифицироваться, поскольку любое изменение (переименование, перемещение, копирование или удаление) любого из них практически гарантированно уничтожит скрытую информацию. Имена и порядок следования покрывающих файлов является частью секретного ключа стеганосистемы.

3. Следует ожидать, что известные методы анализа дискового пространства не смогут обнаружить факт встраивания информационных данных рассмотренными алгоритмами. Это объясняется отсутствием дополнительно внесенной избыточности. Действительно, дополнительная информация на физические носители данных не записывается, изменяется лишь порядок следования отдельных кластеров покрывающих файлов. Тем не менее, фрагментация ФС повышается, повышается также число переплетенных файлов. Однако, как показывают результаты анализа, высокая фрагментация является типовой характеристикой современных ФС, переплетение файлов также встречается достаточно часто. Другими словами, сокрытие данных в кластерных ФС не вносит существенных изменений в

распределение кластеров на физическом носителе и, таким образом, не создает демаскирующих признаков, по которым возможно проведение стеганоанализа.

4. Методы встраивания данных в кластерные ФС является перспективным направлением в развитии современных механизмов стеганографической защиты и могут рассматриваться как реальная альтернатива существующим подходам. Перспективным является проведение дальнейших исследований статистических свойств кластерных ФС, сравнение характеристик фрагментации и переплетения до и после встраивания информационных данных. Эти исследования позволят экспериментально обосновать утверждение об отсутствии демаскирующих признаков рассмотренной стеганосистемы и выработать практические рекомендации по ее дальнейшему использованию.

**Список литературы:** 1. *Hassan Khan, Mobin Javed, Syed Ali Khayam, Fauzan Mirza*. Designing a cluster-based covert channel to evade disk investigation and forensics. *Computers & Security* Volume 30, Issue 1, January 2011. 2. *Hassan Khan, Mobin Javed, Fauzan Mirza*. Evading Disk Investigation and Forensics using a Cluster-Based Covert Channel. National University of Science & Technology (NUST), Islamabad 44000, Pakistan. 3. *Nerijus Morkevičius, Grigas Petraitis, Algimantas Venčkauskas, Jonas Čeponis*. Covert Channel for Cluster-based File Systems Using Multiple Cover Files. *INFORMATION TECHNOLOGY AND CONTROL*, 2013, Vol.42, No.3. 4. *Таненбаум, Э.* Современные операционные системы. – Спб, 2010.

*Харьковский национальный университет  
имени В.Н.Каразина*

*Поступила в редколлегию 23.04.2015*