Эта заявка притязает на приоритет предварительной патентной заявки США № 60/728,089, поданной 18 октября 2005 г., предварительной патентной заявки США № 60/772,024, поданной 9 февраля 2006 г., предварительной патентной заявки США № 60/744,574, поданной 10 апреля 2006 г., предварительной патентной заявки США № 60/791,179, поданной 10 апреля 2006 г., предварительной патентной заявки США № 60/746,712, поданной 8 мая 2006 г., предварительной патентной заявки США № 60/798,925, поданной 8 мая 2006 г., и предварительной патентной заявки США № 60/835,061, поданной 1 августа 2006 г. Предварительные патентные заявки США №№ 60/728,089, 60/772,024, 60/744,574, 60/791,179, 60/746,712, 60/798,925 и 60/835,061 включены сюда посредством ссылки в полном объеме для любых целей.

Определение авторских прав

Уровень техники

В современных вычислительных системах часто бывает желательно ограничивать доступ к ресурсам электронного контента, услуг и/или обработки, и/или разрешать осуществлять определенные действия только определенным сущностям. Для обеспечения такого управления были разработаны или предложены различные методы. Эти методы часто называют методами управления цифровыми правами (DRM), поскольку, в целом, их целью является регулирование прав различных сущностей в отношении цифрового или иного электронного контента, услуг или ресурсов. Проблема, с которой сталкиваются многие методы, отвечающие уровню техники, заключается в том, что они чрезмерно сложны, обременены большим количеством ограничений, сравнительно «негибки», неспособны обеспечивать определенные естественные типы соотношений и процессов, и/или неспособны взаимодействовать с другими системами DRM.

Здесь описаны системы и способы, относящиеся к усовершенствованным методам DRM, которые можно использовать для решения некоторых или всех этих проблем. Очевидно, что варианты осуществления описанного здесь изобретения можно реализовать по-разному, в том числе посредством процессов, аппаратов, систем, устройства, способов, компьютерно-считываемых носителей и/или их комбинации.

Существующие системы для управления доступом к контенту иногда включают в себя компоненты, которые обращаются к лицензии в связи с авторизацией доступа к электронному контенту. Однако такие компоненты обычно осуществляют негибкие вычисления цепей или графов информации управления правами, связей или узлов, связанных с лицензией. Они часто неспособны адаптироваться к различным схемам авторизации и/или работать с определенными системами DRM для авторизации доступа к контенту. Варианты осуществления настоящего изобретения преодолевают такие недостатки за счет сохранения, использования и/или выполнения дополнительных процедур или программ управления в связи с лицензией, что позволяет обеспечивать динамические особенности авторизации, обеспечивать авторизацию распределенных ресурсов и/или другие функции доступа к потоку.

Кроме того, многие существующие системы предназначены только для случаев, когда поддерживаются простые данные, связанные с авторизацией/состоянием. Эти системы неспособны разрешать ситуации, когда для авторизации доступа может потребоваться зависимость от множественных уровней данных, например определение условий на основании ранее выведенных данных, связанных с другими узлами. Варианты осуществления настоящего изобретения преодолевают эти недостатки за счет реализации базы данных состояний совместно с программами управления DRM для обеспечения особенностей хранения состояний, которые являются защищенными, обеспечивают устойчивые информационные состояния от вызова к вызову, или иначе обеспечивают функции чтения и записи состояний, что позволяет улучшать выполнение программ управления и/или осуществлять более эффективную авторизацию доступа.

Дополнительно, существующие системы могут реализовать лицензии или структуры DRM, включающие в себя компоненты, которые предусматривают использование открытых ключей для защиты компонентов лицензии. Однако недостатки этих систем включают в себя возможность того, что хакеры могут подделать цифровые подписи, необходимые для доступа или реализации лицензии, или иначе воспользоваться соответствующими соотношениями, существующими в структуре лицензии DRM. Один или несколько вариантов осуществления настоящего изобретения преодолевают такие недостатки путем реализации цифровой и/или блокирующей подписи объектов «лицензия», включающей в себя использование конкретных защищенных ключей. Преимущества этих вариантов осуществления включают в себя предотвращение неавторизованного доступа посредством открытых ключей, а также соответствующих особенностей, выведенных из соотношений между элементами лицензии.

Другие существующие системы могут включать в себя компоненты, которые могут определять близость между первой сущностью и второй сущностью, например, между двумя сущностями управления правами. Такие системы могут применять правила, указывающий, что фрагмент защищенного контента нельзя копировать за пределы определенной среды, например, путем реализации громоздких процедур

проверки близости. Однако недостатком этих систем является то, что они не способны также предоставлять защиту защищенному контенту без ненужного принудительного осуществления проверки близости. Варианты осуществления настоящего изобретения преодолевают этот и другие недостатки за счет обеспечения лаконичного протокола обнаружения близости, защищенного особенностями, связанными с передачей случайных чисел и/или секретных порождающих чисел. Соответствующие преимущества одного или нескольких вариантов осуществления включают в себя криптографическую нереализуемость для нарушителя определение правильного ответа, даже в случае перехвата запроса.

В итоге, существует необходимость в системах, которые могут адекватно авторизовать доступ к электронному контенту, не прибегая к чрезмерно сложным, обремененным большим количеством ограничений и/или сравнительно негибким методам, в системах, которые способны обеспечивать определенные естественные типы соотношений и процессов, и/или в системах, которые способны взаимодействовать с другими системами DRM.

Раскрытие изобретения

Системы, способы и изделия производства, отвечающие изобретению, относятся к авторизации доступа к электронному контенту и связанной с этим обработке данных.

В одном иллюстративном варианте осуществления, предусмотрен способ авторизации доступа к фрагменту электронного контента, хранящемуся в памяти. Способ, согласно этому иллюстративному варианту осуществления, может включать в себя прием запрос на доступ к электронному контенту, извлечение лицензии, связанной с электронным контентом, и выполнение первой программы управления с использованием механизма управления цифровыми правами для определения, можно ли удовлетворить запрос. Другие иллюстративные варианты осуществления могут включать в себя вычисление одного или нескольких объектов «связь», хранящихся в памяти, и выполнение второй программы управления для определения, действительна(ы) ли связь(и) и/или выполняются ли одно или несколько условий.

Следует понимать, что вышеизложенное общее описание и нижеследующее подробное описание носят исключительно иллюстративный и пояснительный характер и не призваны ограничивать изобретение. Кроме того, могут быть обеспечены признаки и/или вариации помимо изложенных здесь. Например, настоящее изобретение может относиться к различным комбинациям и подкомбинациям раскрытых признаков и/или комбинациям и подкомбинациям некоторых других признаков, раскрытых ниже в подробном описании.

Краткое описание чертежей

Изобретение легче понять, обратившись к нижеследующему подробному описанию, приведенному совместно с прилагаемыми чертежами, в которых:

фиг. 1 - иллюстративная система управления использованием электронного контента;

фиг. 2 - более подробный пример системы, которую можно использовать для практического применения вариантов осуществления изобретения;

фиг. 3 - демонстрирует действие иллюстративного механизма управления цифровыми правами (DRM) в сети, которая использует DRM;

фиг. 4 - совокупность узлов и связей, используемая для моделирования соотношений в системе DRM;

фиг. 5 - логическая блок-схема, демонстрирующая, как вариант осуществления механизма DRM может определять, авторизовано ли запрошенное действие;

фиг. 6 - пример лицензии DRM согласно одному варианту осуществления изобретения;

фиг. 7A и 7B - пример использования агентов в одном варианте осуществления;

фиг. 8 - пример лицензии DRM;

фиг. 9 - более подробный пример того, как механизм DRM может определять, авторизовано ли запрошенное действие;

фиг. 10 - более подробный пример того, как механизм DRM выполняет программу управления в одном варианте осуществления;

фиг. 11 - иллюстративный вариант осуществления механизма DRM, выполняющегося на устройстве;

фиг. 12 - логическая блок-схема, демонстрирующая этапы, предусмотренные при выполнении программы управления в одном варианте осуществления;

фиг. 13 - элементы, составляющие клиентское приложение, потребляющее контент, в одном варианте осуществления;

фиг. 14 - элементы, составляющие приложение упаковки контента в одном варианте осуществления;

фиг. 15 - механизм вывода ключа согласно одному варианту осуществления;

фиг. 16 - пример системы DRM;

фиг. 17 - пример системы DRM, которая обеспечивает временный вход;

фиг. 18 - высокоуровневая архитектура иллюстративной системы для управления документацией предприятия;

фиг. 19 - пример того, как систему, например, показанную на фиг. 18, можно использовать для

управления доступом к документу или другим его использованием;

фиг. 20 - дополнительный пример того, как систему, например, показанную на фиг. 18, можно использовать для управления доступом к документу или другим его использованием;

фиг. 21 - дополнительные особенности примера, показанного на фиг. 20;

фиг. 22 - другая иллюстративная система для управления электронным контентом на предприятии;

фиг. 23 - применение описанных здесь систем и способов для управления записями системы здравоохранения;

фиг. 24 - демонстрирует, как представленные здесь системы и способы можно использовать в контексте службы электронной подписки;

фиг. 25 - демонстрирует, как описанные здесь системы и способы можно использовать в контексте домена домашней сети;

фиг. 26 - взаимодействия, которые имеют место между приложением хоста и механизмом клиента DRM в одном иллюстративном варианте осуществления;

фиг. 27 - взаимодействия, которые имеют место между приложением хоста и механизмом упаковки в одном иллюстративном варианте осуществления;

фиг. 28A - более подробная иллюстрация лицензии согласно одному варианту осуществления;

фиг. 28B - соотношение между связями и узлами в одном иллюстративном варианте осуществления;

фиг. 29 - операционная среда иллюстративной реализации виртуальной машины;

фиг. 30 - структура данных расширенного блока состояний согласно одному варианту осуществления;

фиг. 31A - образ памяти для сегмента данных в одном варианте осуществления;

фиг. 31B - пример образа памяти для сегмента кода в одном варианте осуществления;

фиг. 31C - пример образа памяти для элемента экспорта в одном варианте осуществления;

фиг. 31D - общий пример элемента таблицы экспорта в одном варианте осуществления;

фиг. 31E - пример элемента таблицы экспорта для иллюстративной точки входа;

фиг. 32 - пример протокола передачи лицензии;

фиг. 33 - другой пример протокола передачи лицензии согласно одному варианту осуществления;

фиг. 34 - механизм защиты целостности объектов «лицензия» в одном варианте осуществления;

фиг. 35 - механизм защиты целостности объектов «лицензия» в другом варианте осуществления;

фиг. 36 - протокол проверки близости согласно одному варианту осуществления;

фиг. 37 - демонстрирует использование протокола проверки близости согласно одному варианту осуществления;

фиг. 38 - взаимодействие между клиентом и сервером лицензий в одном варианте осуществления;

фиг. 39 - более подробная иллюстрация взаимодействия между клиентом и сервером лицензий в одном варианте осуществления;

фиг. 40 - пример сущности с множественными ролями;

фиг. 41 - протокол загрузки согласно одному варианту осуществления;

фиг. 42 - соотношение между c14n-ex и иллюстративной XML-каноникализацией в одном варианте осуществления.

Осуществление изобретения

Ниже представлено подробное описание изобретения. Хотя описано несколько вариантов осуществления, следует понимать, что изобретение не ограничивается каким-либо вариантом осуществления, но, напротив, охватывает многочисленные альтернативы, модификации и эквиваленты. Кроме того, хотя в нижеследующем описании приведены многочисленные конкретные детали для обеспечения исчерпывающего понимания изобретения, некоторые варианты осуществления можно применять на практике без некоторых или всех этих деталей. Кроме того, для ясности, определенные технические сведения, известные в уровне техники, не описаны подробно во избежание ненужного затемнения изобретения.

В совместно поданной патентной заявке США № 10/863,551, № 2005/0027871 A1 (далее - "заявке '551"), которая, таким образом, включена сюда посредством ссылки, описаны варианты осуществления архитектуры управление цифровыми правами (DRM) и нового механизма DRM, которые преодолевают некоторые недостатки, характерные для многих предыдущих реализаций DRM. В данной заявке описаны усовершенствования, расширения и модификации, а также альтернативные варианты осуществления архитектуры и механизма DRM, описанных в заявке '551, а также новые компоненты, архитектуры и варианты осуществления. Таким образом, очевидно, что описанный здесь материал можно использовать в контексте архитектуры и/или механизма DRM, например, описанных в заявке '551, а также в других контекстах.

1. Иллюстративная система DRM.

На фиг. 1 показана иллюстративная система 100 для управления электронным контентом. Согласно фиг. 1, сущность 102, обладающая правами на электронный контент 103, упаковывает контент для распространения и потребления конечными пользователями 108a-e (совместно именуемыми "конечными пользователями 108", где позиция 108 в равной степени относится к конечному пользователю и к вычис-

лительной системе конечного пользователя, а к чему именно, будет ясно из контекста). Например, сущность 102 может являться владельцем, создателем или поставщиком контента, например, музыкантом, киностудией, издательством, компанией по разработке программного обеспечения, автором, поставщиком услуг мобильной связи, службой загрузки или подписки на контент в интернете, поставщиком кабельного или спутникового телевещания, служащим корпорации и т.п., или сущностью, действующей от его имени, и контент 103 может содержать любой электронный контент, например цифровой видео-, аудио- или текстовый контент, фильм, песню, видеоигру, фрагмент программного обеспечения, сообщение электронной почты, текстовое сообщение, документ текстового редактора, отчет или любой другой развлекательный, деловой или другой контент.

В примере, показанном на фиг. 1, сущность 102 использует механизм упаковки 109, связывающий лицензию 106 с упакованным контентом 104. Лицензия 106 основана на политиках 105 или других желаниях сущности 102, и указывает разрешенные и/или запрещенные режимы использования контента и/или одно или несколько условий, которым должны удовлетворять режимы использования контента, или которые должны выполняться как условие или следствие использования. Контент также может быть защищен одним или несколькими криптографическими механизмами, например, методами шифрования или цифровой подписи, для чего доверенный орган 110 можно использовать для получения соответствующих криптографических ключей, сертификатов и/или т.п.

Согласно фиг. 1, упакованный контент 104 и лицензии 106 можно предоставлять конечным пользователям 108 любыми пригодными средствами, например, по сети 112 наподобие интернета, локальной сети 103, беспроводной сети, виртуальной частной сети 107, глобальной сети, и/или т.п., посредством кабельной, спутниковой, широковещательной или сотовой связи 114, и/или через записываемые носители 116, например, компакт-диск (CD), цифровой универсальный диск (DVD), карту флэш-памяти (например, карту Security Digital (SD)), и/или т.п. Упакованный контент 104 можно доставлять пользователю совместно с лицензией 106 в одном пакете или передаче 113, или в отдельных пакетах или передачах, принимаемых от одного или разных источников.

Система конечного пользователя (например, персональный компьютер 108e, мобильный телефон 108a, телевизор и/или телевизионная приставка 108c, портативный аудио- и/или видеопроигрыватель, устройство чтения электронных книг и/или т.п.) содержит прикладное программное обеспечение 116, оборудование и/или специализированное логическое устройство, которое способно извлекать и представлять контент. Система пользователя также включает в себя программное обеспечение и/или оборудование, именуемое здесь механизмом 118 управления цифровыми правами, для оценивания лицензии 106, связанной с упакованным контентом 104, и применения ее условий (и/или разрешения приложению 116 применять эти условия), например, путем избирательного предоставления пользователю доступа к контенту только, если это разрешает лицензия 106. Механизм 118 управления цифровыми правами может быть структурно или функционально объединен с приложением 116 или может содержать отдельный фрагмент программного обеспечения и/или оборудования. Альтернативно или дополнительно, система пользователя, например система 108c, может осуществлять связь с удаленной системой, например системой 108b, (например, сервером, другим устройством в сети устройств пользователя, например, персональным компьютером или телевизионной приставкой, и/или т.п.), которое использует механизм управления цифровыми правами для определения 120, предоставлять ли пользователю доступ к контенту, ранее полученному или запрошенному пользователем.

Механизм управления цифровыми правами, и/или другое программное обеспечение, находящееся в системе пользователя или осуществляющее дистанционную связь с ней, также может записывать информацию, относящуюся к доступу пользователя к защищенному контенту или другому его использованию. В некоторых вариантах осуществления, эта информация, полностью или частично, может передаваться удаленной стороне (например, в расчетный центр 122, создателю, владельцу или поставщику контента 102, менеджеру пользователя, сущности, действующей от его имени, и/или т.п.), например, для использования при начислении выручки (например, гонорара, платы за развлечение и т.д.), определении предпочтений пользователя, применении системных политик (например, мониторинге использования конфиденциальной информации), и/или т.п. Очевидно, что, хотя фиг. 1 показана иллюстративная архитектура DRM и совокупность иллюстративных соотношений, описанные здесь системы и способы можно применять на практике в любом подходящем контексте, и, таким образом, очевидно, что фиг. 1 приведена в целях иллюстрации и объяснения, но не в целях ограничения.

На фиг. 2 показан более подробный пример системы 200, которую можно использовать для практического применения вариантов осуществления изобретения. Например, система 200 может содержать вариант осуществления устройства 108 конечного пользователя, устройства 109 поставщика контента и/или т.п. Например, система 200 может содержать вычислительное устройство общего назначения, например, персональный компьютер 108e или сетевой сервер 105, или специализированное вычислительное устройство, например, сотовый телефон 108a, карманный персональный компьютер, портативный аудио- или видеопроигрыватель, телевизионную приставку, киоск, игровую систему и т.п. Система 200 обычно включает в себя процессор 202, память 204, пользовательский интерфейс 206, порт 207, куда вставляется сменная память 208, сетевой интерфейс 210 и одну или несколько шин 212 для соединения

вышеупомянутых элементов. Работой системы 200 обычно управляет процессор 202, действующий под управлением программ, хранящихся в памяти 204. Память 204 обычно включает в себя высокоскоростную оперативную память (ОЗУ) и энергонезависимую память, например, магнитный диск и/или флэш-ЭСППЗУ. Некоторые участки памяти 204 могут быть заблокированы, что не позволяет другим компонентам системы 200 производить на них операции чтения и записи. Порт 207 может содержать дисковод или разъем памяти для приема компьютерно-считываемых носителей 208, например, дискет, CD-ROM, DVD, карт памяти, SD-карт, других магнитных или оптических носителей и/или т.п. Сетевой интерфейс 210 обычно способен обеспечивать соединение между системой 200 и другими вычислительными устройствами (и/или сетями вычислительных устройств) через сеть 220, например, интернет или интрасеть (например, LAN, WAN, VPN и т.д.), и может применять одну или несколько технологий связи для физического создания такого соединения (например, беспроводного, Ethernet, и/или т.п.). В некоторых вариантах осуществления, система 200 также может включать в себя блок обработки 203, защищенный от злонамеренного использования пользователем системы 200 или другими сущностями. Такой защищенный блок обработки может способствовать повышению безопасности важных операций, например, управления ключами, проверки подписи, и других аспектов процесса управления цифровыми правами.

Согласно фиг. 2, память 204 вычислительного устройства 200 может включать в себя различные программы или модули, управляющие работой вычислительного устройства 200. Например, память 204 обычно включает в себя операционную систему 220 для управления выполнением приложений, работой периферийных устройств, и т.п.; приложение хоста 230 для представления защищенного электронного контента; и механизм DRM 232 для реализации некоторых или всех описанных здесь функций управления правами. Как описано здесь в другом месте, механизм DRM 232 может содержать, взаимодействовать с, и/или управлять различными другими модулями, например, виртуальной машиной 222 для выполнения программ управления и базой данных состояний 224 для хранения информации состояния, используемой виртуальной машиной 222, и/или одним или несколькими криптографическими модулями 226 для осуществления криптографических операций, например, шифрования и/или дешифрования контента, вычисления хэш-функций и кодов аутентификации сообщения, оценивания цифровых подписей и/или т.п. Память 204 также обычно включает в себя защищенный контент 228 и соответствующие лицензии 229, а также криптографические ключи, сертификаты и т.п. (не показаны).

Специалисту в данной области техники очевидно, что описанные здесь системы и способы можно применять на практике с использованием вычислительных устройств, аналогичных или идентичных изображенным на фиг. 2, или с использованием практически любого другого подходящего вычислительного устройства, в том числе вычислительного устройства, которое не обрабатывает некоторые из компонентов, показанных на фиг. 2, и/или вычислительного устройства, которое обрабатывает другие компоненты, которые не показаны. Таким образом, очевидно, что фиг. 2 приведена в целях иллюстрации, но не ограничения.

Здесь описаны механизм управления цифровыми правами и соответствующие системы и способы, которые можно использовать для обеспечения некоторых или всех функций управления правами в системах, например, показанных на фиг. 1 и 2, или в системах других типов. Кроме того, ниже описаны различные другие системы и способы, которые можно использовать в контексте систем, например, показанных на фиг. 1 и 2, а также в других контекстах, в том числе, контекстах, не связанных с управлением цифровыми правами.

2. Архитектура механизма DRM.

В одном варианте осуществления, сравнительно простой, открытый и гибкий механизм управления цифровыми правами (DRM) используется для реализации базовых функций DRM. В предпочтительном варианте осуществления, этот механизм DRM должен сравнительно легко интегрироваться в среду веб-услуг, которая, например, описана в заявке '551, и в практически любую среду хоста или архитектуру программного обеспечения. В предпочтительном варианте осуществления, механизм DRM не зависит от конкретных медиа-форматов и криптографических протоколов, что дает разработчикам свободу выбора стандартных или собственных технологий в соответствии с конкретной ситуацией. Предпочтительные варианты осуществления механизма DRM используют простую модель администрирования, но ее можно использовать для выражения сложных соотношений и бизнес-моделей.

Некоторые иллюстративные варианты осуществления механизма DRM, которые описаны ниже, относятся к иллюстративной реализации, именуемой "Octopus"; однако очевидно, что настоящее изобретение не ограничивается конкретными деталями иллюстративного Octopus, которые приведены в целях иллюстрации, но не ограничения.

2.1. Обзор.

На фиг. 3 показано, как иллюстративный механизм DRM 303a может функционировать в системе 302, которая использует DRM.

Согласно фиг. 3, в одном варианте осуществления механизм DRM 303a встроен или интегрирован в приложение хоста 304a (например, приложение представления контента, например, аудио- и/или видеопроигрывателя, приложение представления текста, например, программу электронной почты, текстовый редактор, программу чтения электронных книг или программу чтения документов и/или т.п.) или осуще-

ствляет связь с ним. В одном варианте осуществления, механизм DRM 303a осуществляет функции DRM и опирается на приложение хоста 304a на предмет таких услуг, как шифрование, дешифрование, управление файлами и/или других функций, которые хост может обеспечивать более эффективно. Например, в предпочтительном варианте осуществления, механизм DRM 303a способен манипулировать объектами DRM 305, которые содержат лицензию 306 на защищенный контент 308. В некоторых вариантах осуществления, механизм DRM 303a также может передавать ключи приложению хоста 304a. Согласно фиг. 3, механизм DRM 303a и/или приложение хоста 304a может использовать веб-услуги 305a и/или услуги хоста 306a для обработки и/или информацию, необходимую для решения соответствующих задач. В заявке '551 приведены примеры таких услуг, а также показано, каким образом механизм DRM 303a и приложение хоста 304a могут взаимодействовать с ними.

В примере, показанном на фиг. 3, механизм DRM 303a, приложение хоста 304a, услуги хоста 306a, и интерфейс веб-услуг 305a загружаются в устройство 300a, например персональный компьютер (ПК) конечного пользователя. Устройство 300a поддерживает связь с сервером 300b, от которого поступают контент 308 и лицензия 306, а также с портативным устройством 300d, на которое устройство 300a может пересылать контент 308 и/или лицензию 306. Каждое из этих других устройств может включать в себя механизм DRM 303, аналогичный или идентичный механизму DRM 300a, который может быть интегрирован в конкретное приложение хоста и среду хоста устройства. Например, сервер 300b может включать в себя приложение хоста 304b, которое осуществляет массовую упаковку контента и/или лицензий и использует механизм DRM 300a для оценивания объектов управления, связанных с упаковываемым контентом, для согласования с любыми ограничениями на повторное распространение. Аналогично, устройство 300c может включать в себя приложение хоста 304c, способное представлять и упаковывать контент, тогда как устройство 300a может включать в себя приложение хоста, способное просто представлять контент. В порядке еще одного примера возможного разнообразия сред хоста, устройство 300d может не включать в себя интерфейс веб-услуг, но, вместо этого, может опираться на связь с устройством 300a и интерфейс веб-услуг 305a постольку, поскольку приложение хоста 304d и/или механизм DRM 303d требуют использования каких-либо веб-услуг. На фиг. 3 показан только один пример системы, в которой можно использовать механизм DRM; очевидно, что описанные здесь варианты осуществления механизмов DRM можно реализовать и объединять с приложениями и системами самыми разнообразными способами, не ограничиваясь иллюстративными примерами, показанными на фиг. 3.

2.2. Объекты.

В предпочтительных вариантах осуществления, объекты защиты и администрирования контента используются для представления сущностей в системе, для защиты контента, для связывания правил пользования с контентом и для определения, можно ли предоставить запрошенный доступ.

Как описано более подробно ниже, в одном варианте осуществления, используются следующие объекты:

Тип объекта	Функция
Node (узел)	Представляет сущности
Link (связь)	Представляет направленное соотношение между сущностями
Content (контент)	Представляет контент (например, медиа-контент)
ContentKey (ключ контента)	Представляет ключи шифрования, используемые для шифрования контента
Control (объект управления)	Представляет правила пользования, которые регламентируют взаимодействие с контентом
Controller (контроллер)	Представляет связи между объектами Control и ContentKey
Protector (протектор)	Представляет связи между объектами Content и ContentKey

2.2.1. Объекты «узел».

Объекты «узел» используются для представления сущностей в системе. На практике, узел обычно представляет пользователя, устройство или группу. С объектами «узел» также обычно связаны атрибуты, которые представляют определенные свойства сущности, связанной с узлом.

Например, на фиг. 4 показаны два пользователя (Ксан 400 и Нокс 402), два устройства (ПК 404 и портативное устройство 406), и несколько сущностей, которые представляют группы (например, членов семьи Кери 408, абонентов публичной библиотеки 410, подписчиков на конкретные музыкальные услуги

412, устройств 414, одобренных RIAA (Американской ассоциацией звукозаписывающих компаний), и устройств 416, изготовленных конкретной компанией), с каждой из которых связан объект «узел».

В одном варианте осуществления объекты «узел» включают в себя атрибуты, указывающие, что представляет узел. Один пример атрибута это тип узла. Помимо представления пользователей, групп или устройств, атрибут «тип узла» можно использовать для представления других сущностей. В некоторых вариантах осуществления, объект «узел» также может включать в себя информацию криптографического ключа, например, при использовании варианта осуществления методов вывода и распространения ключа, описанных здесь в другом месте.

В некоторых вариантах осуществления, объекты «узел» также включают в себя пару асимметричных ключей конфиденциальности, которая используется для «нацеливания» конфиденциальной информации на подсистемы, имеющие доступ к конфиденциальным частям объекта «узел». Это может быть сущность, которую представляет узел (например, Музыкальные услуги 412) или некоторая сущность, отвечающая за управление узлом (например, конечный пользователь (например, Нокс 402) может отвечать за управление своим портативным устройством 406).

2.2.2. Объекты «связь».

В предпочтительном варианте осуществления, объекты «связь» представляют собой подписываемые объекты, используемые для показа соотношения между двумя узлами. Например, согласно фиг. 4, связь 418 от узла ПК 404 к узлу Нокс 402 показывает право собственности. Связь от узла Нокс 402 к узлу «семья Кери» 408 показывает принадлежность, как и связь от узла «семья Кери» 408 к узлу «подписчики на музыкальные услуги» 412. В одном варианте осуществления, объекты «связь» выражают соотношение между двумя узлами, и, таким образом, соотношения, показанные на фиг. 4, можно представить с использованием десяти связей.

Согласно фиг. 4, граф 420 можно использовать для выражения соотношения между узлами, где объекты «связь» являются ориентированными ребрами между узлами. Например, на фиг. 4, соотношение между узлом «семья Кери» 408 и узлом «музыкальные услуги» 412 указывает, что существует ориентированное ребро 422 в графе, вершинами которого являются узел «семья Кери» 408 и узел «музыкальные услуги» 412. Нокс 402 и Ксан 400 являются членами семьи Кери 408. Поскольку Нокс 402 связан с семьей Кери 408, и семья Кери 4 08 связана с Музыкальными услугами 412, должен быть путь между Ноксом 402 и Музыкальными услугами 412. Механизм DRM рассматривает узел как доступный из другого узла, когда существует путь от этого узла к другому узлу. Это позволяет записать управляющий элемент, который позволяет разрешать доступ к защищенному контенту на основании того условия, что узел доступен из устройства, на котором выполняется приложение, которое запрашивает доступ к защищенному контенту.

Как описано более подробно ниже, объекты «связь» также могут, в необязательном порядке, содержать некоторые криптографические данные, которые позволяют выводить ключи контента. Объекты «связь» также могут содержать программы управления, которые задают условия, при которых связь можно считать действительной. Такие программы управления могут выполняться или интерпретироваться (эти термины используются здесь взаимозаменяемо) виртуальной машиной механизма DRM для оценивания действительности связи (например, для определения, можно ли использовать связь для достижения данного узла в графе авторизации).

В одном варианте осуществления, связи являются подписываемыми. Можно использовать любой подходящий механизм цифровой подписи, и в одном варианте осуществления механизм DRM не задает, как подписываются объекты «связь», и не оценивает никакие соответствующие сертификаты, вместо этого, он опирается на систему хоста для проверки любых таких подписей и/или сертификатов. Это позволяет архитектору или администратору системы задавать срок действия объекта «связь», отменять его и т.д. (например, с использованием истечения срока действия, отмены и/или т.п. ключей или сертификатов), тем самым обеспечивая дополнительный уровень управления политикой и безопасности на вершине управления политикой и безопасности, обеспечиваемых оцениванием программ управления и объектов DRM механизмом DRM в контексте конкретных фрагментов защищенного контента и/или связей (например, истечение срока действия связи можно, альтернативно или дополнительно, реализовать путем включения соответствующей программы управления в сам объект «связь», которая, при выполнении, будет применять дату окончания срока действия или другой период действия). В одном варианте осуществления, механизм DRM является общим и работает с любой подходящей схемой шифрования, цифровой подписи, отмены и/или другой схемой безопасности, которая используется приложением и/или средой хоста. Таким образом, например, если механизму DRM нужно определить, имеет ли конкретная связь надлежащую подпись, он может просто вызвать приложение хоста (и/или криптографическую службу хоста или системы) для проверки подписи в соответствии с конкретной схемой подписи, выбранной проектировщиком системы, детали которой могут быть неизвестны механизму DRM. В других вариантах осуществления, механизм DRM сам осуществляет фактическое оценивание подписи, с опорой на хост, просто для указания использования надлежащего алгоритма подписи.

2.2.3. Защита и администрирование контента.

Согласно фиг. 3, в типичном случае, поставщик контента 300b использует приложение 304b, кото-

рое включает в себя механизм упаковки для шифрования или иной криптографической защиты фрагмента электронного контента 308, и создает лицензию 306, которая регламентирует доступ к этому контенту или другое его использование. В одном варианте осуществления, лицензия 308 содержит набор объектов 305, которые указывают, как можно использовать контент 308, а также включает в себя ключ(и) шифрования контента и/или информацию, необходимую для его(их) получения. В одном варианте осуществления, контент 308 и лицензия 306 логически разделены, но связаны друг с другом внутренними ссылками (например, с использованием ID объекта 310). Во многих случаях, удобно сохранять и/или доставлять контент и лицензию совместно; однако в предпочтительных вариантах осуществления это не требуется. В одном варианте осуществления, лицензию можно применять к более чем одному элементу контента, и более чем одну лицензию можно применять к любому отдельно взятому элементу контента.

Согласно фиг. 3, когда приложение хоста 304a, выполняющееся на клиентском устройстве 300a, хочет осуществить действие на конкретном фрагменте контента 308, оно просит механизм DRM 303a проверить, разрешено ли действие, которое оно намеревается совершить (например, "воспроизведение"). В одном варианте осуществления, механизм DRM 303a, из информации, содержащейся в объектах 305, содержащих лицензию контента 306, загружает и выполняет программу управления, связанную с контентом 308, и разрешение на осуществление действия дается или не дается на основании результата, возвращенного программой управления. Для дачи разрешения обычно требуется выполнение некоторых условий, например, условия доступности узла из узла, представляющего запрашивающую(ее) сущность/устройство 300a.

На фиг. 5 показана логическая блок-схема, демонстрирующая, как механизм DRM, согласно варианту осуществления, может определять, авторизовано ли запрошенное действие (например, просмотр фрагмента контента). Согласно фиг. 5, принимается (500) запрос на оценивание лицензии на данное действие. Например, этот запрос может поступать от приложения хоста, после того, как хост принимает от пользователя запрос на осуществление указанного действия. Согласно фиг. 5, механизм DRM оценивает указанную лицензию (502) и определяет, авторизовано ли запрошенное действие (504). Например, лицензия может содержать программу управления, которую выполняет механизм DRM, выход которой используется для принятия решения на авторизацию. Если лицензия авторизует запрошенное действие (т.е. на выходе "да" блока 504), то механизм DRM указывает приложению хоста, что запрос удовлетворен (506). В противном случае, механизм DRM указывает приложению хоста, что запрос отклонен (508). В некоторых вариантах осуществления, механизм DRM может также возвращать приложению хоста различные метаданные, которые, например, связывают условия с предоставлением авторизации (например, обязательства и/или обратные вызовы) или обеспечивают дополнительную информацию, касающуюся причины отказа в авторизации. Например, механизм DRM может указать, что запрошенное действие разрешено только, если приложение хоста регистрирует определенную информацию, относящуюся к осуществлению запрошенного действия, или при условии, что приложение хоста осуществляет обратные вызовы механизма DRM с заранее заданными интервалами времени, например, для повторного оценивания лицензии. Дополнительная информация о таких обязательствах, обратных вызовах, и другие метаданные, возвращаемые механизмом DRM, описаны ниже. Если запрошенное действие авторизовано, ключ контента извлекается (например, из объекта ContentKey лицензии) и используется для отпуска контента для запрашиваемого использования.

2.2.4. Объекты DRM лицензии.

Согласно фиг. 6, в предпочтительном варианте осуществления лицензия 600 представляет собой совокупность объектов. В примере, показанном на фиг. 6, лицензия 600 содержит объект ContentKey 602, объект «протектор» 604, объект «контороллер» 606, и объект управления 608. Согласно фиг. 6, объект ContentKey 602 включает в себя зашифрованные данные ключа 610 (например, зашифрованную версию ключа, необходимого для дешифрования зашифрованного элемента контента 612) и информацию, относящуюся к криптосистеме, используемой для шифрования данных ключа. Объект «протектор» 604 связывает объект ContentKey 602 с одним или несколькими объектами «контент» 614. Согласно фиг. 6, объект управления 608 включает в себя и защищает программу управления 616, которая указывает, как регламентируется объект «контент» 614. В предпочтительном варианте осуществления, программа управления 616 является фрагментом исполнимого байт-кода, который выполняется на виртуальной машине, используемой механизмом DRM. Программа управления определяет, можно ли осуществлять определенные действия на контенте, проверяя выполнение условий, указанных в программе управления, например, доступны ли определенные узлы с использованием действительных объектов «связь», сохранены ли определенные объекты состояния, имеет ли среда хоста определенные характеристики, и/или т.п. Согласно фиг. 6, объект «контроллер» 606 используется для связывания одного или нескольких объектов ContentKey 602 с объектом управления 608.

Лицензия 600 также может содержать дополнительные объекты, например, метаданные, обеспечивающие описание, считываемое машиной или человеком, условий доступа к контенту, требуемых лицензией. Альтернативно или дополнительно, такие метаданные могут быть включены в качестве расширения ресурсов одного из других объектов (например, объекта управления 608). Согласно варианту осуществления, показанному на фиг. 6, объект управления 608 и объект «контроллер» 606 являются подписы-

ваемыми, что позволяет системе удостовериться в том, что информация управления поступила из доверенного источника, прежде чем использовать ее для принятия решений относительно доступа к контенту. В одном варианте осуществления, действительность объекта управления 608 также можно проверять путем проверки защищенного хэша, включенного в объект «контроллер» 606. Объект «контроллер» 606 также может содержать значение хэша для каждого из ключей или других данных ключа, содержащихся в объекте(ах) ContentKey 602, на который(е) он ссылается, благодаря чему нарушителю довольно трудно подделать его представление путем установления связи между данными ключа и объектом ContentKey.

Согласно фиг. 6, в одном варианте осуществления контент 612 зашифрован и включен в объект «контент» 614. Используемый ключ дешифрования 610 включен в объект ContentKey 602 (или представлен им), и связь между ними представлена объектом «протектор» 604. Согласно фиг. 6, уникальные ID используются для облегчения установлению связи объектом «контент» 614 и объектом ContentKey 602. Правила, которые регламентируют использование ключа 610 для дешифрования контента 612, включены в объект управления 608, и связь между объектом управления 608 и объектом ContentKey 602 представлена объектом «контроллер» 606, опять же, с использованием уникальных ID.

Очевидно, что, хотя на фиг. 6 показаны объекты, содержащие лицензию в одном предпочтительном варианте осуществления, описанные здесь системы и способы DRM не ограничиваются использованием этой структуры лицензии. Например, без ограничения, можно использовать лицензии, в которых функции различных объектов, показанных на фиг. 6, объединены в меньшее количество объектов, или распределены по дополнительным объектам, или разбиты между объектами иным образом. Альтернативно или дополнительно, варианты осуществления описанных здесь систем и способов можно применять на практике с лицензиями, которым недостает некоторых функций, обеспечиваемых структурой лицензии, показанной на фиг. 6, и/или, в которых предусмотрены дополнительные функции. Таким образом, очевидно, что можно использовать любой подходящий механизм для связывания лицензий с контентом в соответствии с описанными здесь принципами, хотя в предпочтительных вариантах осуществления используется преимущественная структура, показанная на фиг. 6.

2.3. База данных состояний.

В одном варианте осуществления, механизм DRM включает в себя защищенное постоянное хранилище объектов, или имеет доступ к нему, которое можно использовать для обеспечения механизма защищенного хранилища состояний. Такое приспособление полезно для обеспечения программ управления, способных считывать и записывать информацию состояния, которая сохраняется от вызова к вызову. Такую базу данных состояний можно использовать для сохранения объектов состояния, например счетчиков воспроизведения, даты первого использования, совокупного времени представления и/или т.п., а также состояния принадлежности и/или любых других пригодных данных. В некоторых вариантах осуществления, механизм DRM, выполняющийся на первой системе, может не иметь доступа к локальной базе данных состояний, и может иметь возможность обращаться к удаленной базе данных состояний, например, с использованием веб-услуг и/или услуг хоста. В ряде случаев, механизму DRM, выполняющемуся на первой системе, может понадобиться доступ к информации состояния, хранящейся в базе данных на удаленной системе. Например, первая система может не включать в себя базу данных состояний, или может не иметь информации, в которой она нуждается, в своей собственной базе данных состояний. В некоторых вариантах осуществления, когда механизм DRM сталкивается с подобной ситуацией, он может обращаться к удаленной базе данных состояний через интерфейс услуг, и/или с использованием программ-агентов, что описано более подробно ниже.

2.4. О программах управления.

Описанные здесь системы и способы используют программы управления в различных контекстах. Например, программы управления, содержащиеся в объектах управления, можно использовать для выражения правил и условий, регламентирующих использование защищенного контента. Кроме того, программы управления в объектах «связь» можно использовать для выражения правил и условий, используемых для определения, пригодна ли связь для данной цели (например, анализа доступности узла). Такие программы управления иногда называются здесь ограничениями по связи. Еще один контекст, в котором можно использовать программы управления, это объекты «агент» или «делегат», где код управления используется для осуществления действия от имени другой сущности (в случае агентских программ управления) или от имени другого объекта управления (в случае делегатских программ управления).

В одном варианте осуществления, программы управления выполняются или интерпретируются виртуальной машиной, базирующейся на механизме DRM, а не выполняющейся непосредственно физическим процессором. Однако очевидно, что можно легко построить физический процессор или иное логическое устройство для выполнения программ управления. В одном варианте осуществления, программы управления имеют формат байт-кода, что дает дополнительную возможность взаимодействия между платформами.

В предпочтительном варианте осуществления, программы управления написаны на языке ассемблера и преобразованы в байт-код программой ассемблера. В других вариантах осуществления можно использовать шаблоны и/или языки выражения прав высокого уровня для обеспечения начального выражения прав, правил и/или условий, и можно использовать компилятор для преобразования выражения

высокого уровня в байт-код для выполнения механизмом DRM согласно описанному здесь варианту осуществления. Например, выражения прав, записанные в собственном формате DRM, можно, с помощью соответствующего компилятора, преобразовать или транслировать в функционально эквивалентное выражение на уровне байт-кода для выполнения на механизме DRM согласно описанному здесь варианту осуществления, что позволяет использовать защищенный фрагмент контента, в соответствии с условиями, указанными поставщиком контента, на системах, которые понимают собственный формат DRM, а также системах, которые включают в себя механизм DRM, например, описанный здесь. Также очевидно, что описанные здесь системы и способы на основе механизма управления цифровыми правами не ограничиваются использованием выражений прав в формате байт-кода, интерпретируемых виртуальной машиной. Напротив, в некоторых вариантах осуществления, права можно выражать любым подходящим способом (например, с использованием языка выражения прав высокого уровня (REL), шаблона, и т.д.), и граф авторизации и/или другие описанные здесь методы, осуществляемые с использованием прикладной программы, предназначенной для распознавания и оценивания таких выражений прав.

2.4.1. Условия.

Как указано выше, программы управления обычно выражают одно или несколько условий, которые должны выполняться для удовлетворения запроса на использование фрагмента контента, для того, чтобы связь была признана действительной, и/или т.п. Можно использовать любые подходящие условия, в зависимости от требований поставщика контента или архитектора системы, и/или функций, обеспечиваемых системой.

В предпочтительных вариантах осуществления, виртуальная машина, используемая механизмом DRM, поддерживает программы любой сложности, которые способны проверять выполнение условий, например некоторые или все из следующих:

Условия на основе времени: сравнение значения времени клиента со значением или значениями, указанным(и) в программе управления.

Нацеливание конкретного узла: проверка, доступен ли определенный узел из другого узла. Эта концепция обеспечивает поддержку таких моделей, как домены, подписки, отношения принадлежности и т.п.

Проверка, совпадают ли атрибуты определенного узла с указанными значениями: проверка любых атрибутов узла, например, удовлетворяют ли возможности представления устройства, связанного с узлом, требованиям верности.

Проверка, обновлены ли метаданные, связанные с безопасностью, на клиенте: проверка, например, имеет ли клиент приемлемую версию клиентского программного обеспечения, и точное измерение времени. В некотором варианте осуществления, такая проверка может опираться, например, на утверждения в одном или нескольких сертификатах от службы сертификации данных.

Условия на основе состояния: проверка информации в базе данных состояний. Например, база данных состояний может содержать информацию, сгенерированную в результате предыдущего выполнения программ управления, и/или жетоны, свидетельствующие о владении подписками, принадлежности и/или т.п., что позволяет оценивать условия, определяемые счетчиками (например, количество актов воспроизведения, количество актов экспорта, пределы истекшего времени и т.д.) и другую информацию, относящуюся к зарегистрированным событиям и условиям.

Характеристики среды. Например, проверка, имеет ли оборудование и/или программное обеспечение в среде хоста определенные характеристики, например, способность распознавать и применять обязательства; проверка наличия или отсутствия определенных программных или аппаратных компонентов, например, защищенного выходного канала; проверка информации близости, например, близости запрашивающего устройства к другому устройству или приложению; проверка характеристик удаленных систем и/или хранящихся в них данных, с использованием сетевых услуг и/или агентов; и/или т.п.

С использованием этих или любых других подходящих условий, объект управления может выражать правила, которые регламентируют представление, перенос, экспорт и/или т.п. контента. Очевидно, что вышеприведенный список условий носит иллюстративный характер, и что можно задать и использовать любые подходящие условия, например, путем реализации системного вызова для использования при проверке выполнения нужного условия. Например, без ограничения, если желательно потребовать, чтобы устройство находилось в конкретной подсети, можно задать системный вызов (например, GetIPConfig), который способен возвращать информацию IPConfig устройства хоста (или информацию IPConfig удаленного устройства, если системный вызов запущен на удаленном устройстве с использованием агента), который может использовать программа управления для проверки, находится ли устройство в предписанной подсети.

2.4.2. Агенты.

Предпочтительные варианты осуществления описанных здесь систем и способов на основе механизма DRM обеспечивают поддержку независимых объектов, несущих программы управления. Такие "агенты" могут распространяться на механизм DRM, выполняющийся на удаленной системе, для выполнения указанных функций, например, записи в защищенное хранилище состояний удаленного механизма DRM. Например, агент может передаваться вследствие контакта с удаленной службой или выполнения

удаленной программы управления. Агент также можно использовать для осуществления операции перемещения контента, для инициализации счетчика, для отмены регистрации узла и/или т.п. В порядке еще одного примера, агент можно использовать для осуществления анализа доступности из удаленного узла к другому узлу. Такой агент, например, может быть полезен при применении политики, запрещающей второму пользователю регистрировать устройство, зарегистрированное первым пользователем. Если второй пользователь запросил регистрацию, агент может быть направлен на устройство вторым пользователем или службой регистрации, действующей от его имени, для определения, было ли устройство ранее зарегистрировано для первого пользователя, в каковом случае запрос второго пользователя на регистрацию будет отклонен.

На фиг. 7А и 7В показано использование агентов в одном варианте осуществления. Согласно фиг. 7А, предположим, что две сущности - система А 700 и система В 702 - желают связаться друг с другом через компьютерную сеть 703, и что используется система DRM, способная описывать и применять правила для определенных операций, например, доступа к защищенному контенту или создания объектов DRM, которые можно использовать для представления отношений принадлежности, состояния регистрации и/или т.п. в ряде случаев, правило(а) оцениваются на системе А 700, но для этого требуется информация, которая зависит от состояния системы В 702. Система DRM 704, которая применяет правило (а) на системе А 700, должна доверять этой информации.

Например, система DRM 704 на системе А 700 может оценивать/применять правило осуществления дистанционного представления контента из системы А 700 в системе В 702, и правило может указывать, что такая операция разрешена только, если система В 702 входит в состав определенной группы устройств, причем принадлежность к этой группе подтверждается наличием объекта «состояние» 711 в защищенной базе данных состояний 716, доступной на системе В 702.

Способ, используемый в предпочтительном варианте осуществления для работы в таких ситуациях, использует агенты. Например, если системе А 700 нужна информация из системы В 702, система А 700 подготавливает агент 705, который, в одном варианте осуществления, является программой управления (например, последовательностью команд, которые могут выполняться механизмом DRM), которая передается из системы А 700 в систему В 702. В одном варианте осуществления, система А 700 передает агентский код 705 в систему В 702 по аутентифицированному каналу связи 720, благодаря чему система А 700 может быть уверена, что агент 705 будет выполняться именно в системе В 702. В некоторых вариантах осуществления, помимо агентского кода 705, система А 700 также может передавать системе В 702 один или несколько параметров, которые агентский код 705 может использовать для осуществления своей работы.

Согласно фиг. 7В, система В 702 принимает агент 705 и любые параметры, связанные с агентом, и запускает агентский код 705. Когда агент 705 выполняется на системе В 702, он обращается к базе данных состояний 716 системы В, извлекает информацию состояния 711 и/или осуществляет одно или несколько вычислений над ней, и передает результаты 713 обратно в систему А 700, предпочтительно по аутентифицированному каналу связи 710, благодаря этому, система А 700 имеет информацию, которая ей нужна для продолжения своего оценивания.

2.4.3. Ограничения по связи.

В одном варианте осуществления, набор процедур, которые представляют правила, регламентирующие осуществление определенной операции (например "воспроизведение") на элементе контента, называется "управляющий элемент действия". Набор процедур, которые представляют ограничения по действительности на объекте «связь» называется "ограничение по связи". Наподобие управляющих элементов действия, в предпочтительных вариантах осуществления ограничения по связи могут выражать любую подходящую комбинацию условий. Также аналогично управляющим элементам действия, ограничения по связи можно оценивать локально и/или дистанционно с использованием интерфейса услуг или агента.

2.4.4. Обязательства и обратные вызовы.

В одном варианте осуществления, определенные действия, когда они разрешены, требуют дополнительного участия приложения хоста. Обязательства представляют операции, которые должно осуществлять приложение хоста после использования ключа контента, которые они запрашивают. Обратные вызовы представляют вызовы одной или нескольких процедур программы управления, которые должно осуществлять приложение хоста после использования ключа контента, которые они запрашивают. Примеры обязательств включают в себя, без ограничения, требование, чтобы определенные выходы и/или управляющие элементы были отключены в ходе представления контента (например, для предотвращения записи контента в незащищенный выход или для предотвращения перемотки вперед определенных важных сегментов контента); требование, чтобы информация, относящаяся к использованию контента записывалась (например, информация измерения или аудита) и/или передавалась в удаленное место (например, в расчетный центр, поставщику услуг и т.п.); требование, чтобы программа-агент выполнялась локально или дистанционно; и/или т.п. Примеры обратных вызовов включают в себя, без ограничения требование, чтобы хост осуществлял обратный вызов программы управления в определенное абсолютное время, по истечении определенного времени (например, времени пользования контентом), по наступле-

нии определенного события (например, по завершении периода пробного представления контента), когда использование контента остановлено, и/или т.п. Например, обратный вызов по истечении определенного времени можно использовать для увеличения или уменьшения бюджетов, счетчиков воспроизведения и т.п. (например, дебетования бюджета пользователей только, если они используют фрагмент контента в течение, по меньшей мере, определенного времени), что защищает пользователя от списания с его лицевого счета в случае, если он непреднамеренного нажмет кнопку воспроизведения, но сразу же нажмет кнопку остановки.

В одном варианте осуществления, существуют разные типы обязательств и обратных вызовов, и если приложение сталкивается с каким-либо критическим обязательством или обратным вызовом, которое(ый) он не поддерживает или не понимает (например, потому, что тип обязательства был задан после реализации приложения), приложение должно отказаться от продолжения действия, для которого было возвращено это(т) обязательство или параметр обратного вызова.

2.4.5. Пример.

На фиг. 8-12 показан пример того, как иллюстративный вариант осуществления механизма DRM может управлять использованием фрагмента контента. Согласно фиг. 8, предположим, что механизм DRM принял запрос на воспроизведение группы 800 элементов контента 802, 804. Например, элементы контента 802, 804 могут содержать разные подчасти мультимедиа-представления, различные треки альбома, различные фрагменты контента, полученные от службы подписки, вложенные файлы электронной почты и т.п. Запрос может приниматься механизмом DRM от приложения хоста, которое, в свою очередь, приняло запрос от пользователя вычислительного устройства, после чего приложение хоста было выполнено. Запрос от приложения хоста обычно идентифицирует запрошенное действие, фрагмент или фрагменты контента, после которых должно быть выполнено действие, и лицензию(и), которая(ые) управляют контентом. Механизм DRM выполняет процесс, показанный на фиг. 5, для определения, нужно ли удовлетворить запрос.

На фиг. 8 и 9 обеспечен более подробный неограничительный пример процесса, показанного на фиг. 5. Согласно фиг. 9, после приема запроса на доступ к элементам контента 802 и 804 (блок 900), механизм DRM проверяет лицензию(и), указанные в запросе, или иначе связанную(ые) с ним, чтобы определить, существует ли действительная лицензия. Например, механизм DRM может сначала идентифицировать объекты «протектор» 806 и 808, которые содержат уникальные идентификаторы элементов контента 802 и 804 (т.е. NS:007 и NS:008, соответственно) (блок 902 на фиг. 9). Затем, механизм DRM обнаруживает объекты ContentKey 810 и 812, указанные в объектах «протектор» 806 и 808 (блок 904 на фиг. 9), что, в свою очередь, позволяет механизму DRM идентифицировать контроллер 814, который ссылается на оба объекта ContentKey 810 и 812 (блок 906 на фиг. 9). В предпочтительном варианте осуществления, контроллер 814 подписан, и механизм DRM проверяет его подпись (или запрашивает услуги хоста для ее проверки). Механизм DRM использует контроллер 814 для идентификации объекта управления 816, который регламентирует использование объекты ContentKey 810 и 812 (и, таким образом, элементы контента 802 и 804) (блок 908 на фиг. 9). В предпочтительном варианте осуществления, механизм DRM проверяет целостность объекта управления 816 (например, путем вычисления дайджеста объекта управления 816 и сравнения его с дайджестом, содержащимся в контроллере 814). Если проверка целостности успешна, механизм DRM выполняет код управления, содержащийся в объекте управления 816 (блок 910), и возвращает результат (блок 912) приложению хоста, которое использует его для удовлетворения или отклонения запроса пользователя на доступ к контенту. Результат кода управления также может, в необязательном порядке, указывать одно или несколько обязательств или обратных вызовов, которые понадобиться выполнить приложению хоста.

На фиг. 10 показан более подробный пример того, как механизм DRM может осуществлять действия, указанные в блоках 910 и 912 на фиг. 9 (т.е. выполнять программу управления и возвращать результат). Согласно фиг. 10, идентифицировав соответствующий объект управления, механизм DRM загружает байт-код, содержащийся в объекте управления, в виртуальную машину, которая, предпочтительно, базируется на механизме DRM (блок 1000). Механизм DRM и/или виртуальная машина также обычно инициализирует среду выполнения виртуальной машины (блок 1002). Например, виртуальная машина может выделять память, необходимую для выполнения программы управления, инициализировать регистры и другие переменные среды, и/или получать информацию о среде хоста, в которой действует виртуальная машина (например, делая вызов System.Host.GetObject, описанный ниже). Очевидно, что в некоторых вариантах осуществления блоки 1000 и 1002 можно эффективно комбинировать или перемежать и/или обращать их порядок. Согласно фиг. 10, виртуальная машина затем выполняет байт-код программы управления (блок 1004). Как описано здесь в другом месте, для этого можно делать вызовы кода другой виртуальной машины, извлекать информацию состояния из защищенного хранилища и/или т.п. По окончании выполнения программы управления, она обеспечивает выход (например, в предпочтительном варианте осуществления, ExtendedStatusBlock), который может использовать, например, вызывающее приложение для определения, был ли удовлетворен запрос, и, если это так, связаны ли с ним какие-либо обязательства или обратные вызовы; был ли отклонен запрос, и, если это так, по какой причине; или произошли ли какие-либо ошибки в ходе выполнения (блок 1006).

Как указано выше, код управления, содержащийся в объекте управления 816, указывает условия или другие требования, которые должны быть выполнены для осуществления запрошенного использования элементов контента 802 и 804. Описанные здесь системы и способы позволяют задавать наборы условий любой сложности; однако, в целях этого примера, предположим, что программа управления призвана требовать, чтобы, для воспроизведения элементов контента 802 и 804, (a) данный узел пользователя был доступен из устройства, на котором был сделан запрос на воспроизведение контента, и (b) текущая дата была позже указанной даты.

На фиг. 11 показано, как иллюстративный вариант осуществления механизма DRM 1100, выполняющийся на устройстве 1102, может выполнять вышеописанную иллюстративную программу управления, и на фиг. 12 показана логическая блок-схема этапов, предусмотренных при выполнении процесса. Согласно фиг. 11, механизм DRM 1100 создает контекст выполнения виртуальной машины (например, путем вызова System.Host.SpawnVm) 1104 и загружает программу управления. Виртуальная машина 1104 начинает выполнение программы управления в точке входа, указанной механизмом DRM 1100 (например, в положении процедуры Control.Actions.Play.perform). В этом примере, программа управления нуждается в определении, доступен ли данный узел из узла индивидуальных особенностей устройства 1102, на котором выполняется механизм DRM 1100. Чтобы произвести такое определение, программа управления вызывает 1105 услугу менеджера связей 1106, предоставляемую механизмом DRM 1100, указывая узел, с которым требуется установить связь (блок 1200 на фиг. 12). Менеджер связей 1106 отвечает за оценивание объектов «связь» для определения, доступен ли один узел из другого. Чтобы делать это эффективно, менеджер связей 1106 может предварительно вычислять, существует ли путь от узла индивидуальных особенностей 1110 устройства 1102 к различным узлам 1114, указанным в любых объектах «связь», которыми обладает это устройство 1102. Таким образом, менеджер связей 1106 может, просто проверяя поля "к" и "от" связей, к которым он обращается, определять, какие узлы потенциально доступны из узла индивидуальных особенностей 1110 устройства 1102. Когда менеджер связей 1106 принимает вызов 1105 от виртуальной машины 1104, он определяет, доступен ли указанный узел 1112, сначала определяя, существует ли путь от узла индивидуальных особенностей 1110 к указанному узлу 1112 (например, проверяя ID узла в списке узлов, которые ранее были определены как теоретически доступные) (блок 1202 на фиг. 12). Если путь существует, менеджер связей 1106 оценивает любые программы управления, содержащиеся в связях, чтобы определить, действительны ли связи (блоки 1204-1210 на фиг. 12). Чтобы оценить программы управления в объектах «связь» (блок 1206 на фиг. 12), менеджер связей 1106 может использовать свою собственную виртуальную машину 1108, на которой он выполняет программы управления, включенные в объекты «связь». Менеджер связей 1106 возвращает результаты своего определения (т.е. доступен ли данный узел) программе управления, выполняющейся на виртуальной машине 1104, где она используется для общего оценивания, будет ли удовлетворен запрос на воспроизведение фрагмента контента. После определения, что указанный узел 1112 доступен из узла индивидуальных особенностей 1110 устройства 1102, программа управления, выполняющаяся на виртуальной машине 1104, определяет, выполняется ли указанное ограничение по дате (блок 1212 на фиг. 12). Если ограничение по дате выполнено (т.е. на выходе "да" из блока 1212), то программа управления возвращает результат, указывающий, что указанные условия выполнены (блок 1214 на фиг. 12); в противном случае, программа управления возвращает результат, указывающий, что указанные условия не выполнены (блок 1216 на фиг. 12).

Пример программы управления, например вышеописанной, показан ниже.

```
; Иллюстративная программа управления
;
; Эта программа управления проверяет, что узел «пользователь» доступен
; и что дата позже конкретной начальной даты
; и раньше конкретной конечной даты
; Значения извлекаются из атрибутов в программе управления

; ==========================
; константы
; ==========================
   .equ DEBUG_PRINT_SYSCALL,                1
   .equ FIND_SYSCALL_BY_NAME,               2
   .equ SYSTEM_HOST_GET_OBJECT_SYSCALL,     3
   .equ SUCCESS,                            0
   .equ FAILURE,                           -1

; ==========================
; данные
; ==========================
.data

ControlTargetNodeIdAttributePath:
   .string "Octopus/Control/Attributes/TargetNodeId"
ControlStartDateAttributePath:
   .string "Octopus/Control/Attributes/StartDate"
ControlEndDateAttributePath:
   .string "Octopus/Control/Attributes/EndDate"
TargetNodeId:
    .zeros 256
StartDate:
   .long    0
EndDate:
   .long   -1
IsNodeReachableFunctionName:
   .string "Octopus.Связи.IsNodeReachable"
```

```
IsNodeReachableFunctionNumber:
   .long   0
GetTimeStampFunctionName:
   .string "System.Host.GetLocalTime"
GetTimeStampFunctionNumber:
   .long   0

; ========================
; код
; ========================
.code

Global.OnLoad:
; загрузка глобальных функций

   ; получить номер системного вызова для Octopus.Links.IsNodeReachable
   PUSH @IsNodeReachableFunctionName
   PUSH FIND_SYSCALL_BY_NAME
   CALL
   DUP
   PUSH @IsNodeReachableFunctionNumber
   POKE
   BRN OnLoad_Fail

   ; получить номер системного вызова для System.Host.GetTimeStamp
   PUSH @GetTimeStampFunctionName
   PUSH FIND_SYSCALL_BY_NAME
   CALL
   DUP
   PUSH @GetTimeStampFunctionNumber
   POKE
   BRN OnLoad_Fail

   ; ok
   PUSH    0
   RET

OnLoad_Fail:
   PUSH FAILURE
   RET

Control.Actions.Play.Init:
   ; получить значения из атрибутов
   ; получить конечный узел (гарантированно существует)
   PUSH    256                                    ; ReturnBufferSize (256 байт)
   PUSH @TargetNodeId                             ; Возвращаемое значение
   PUSH @ControlTargetNodeIdAttributePath         ; Имя
   PUSH 0                                         ; Родитель = корневой контейнер
   PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
   CALL

   ; получить начальную дату
```

```
  PUSH 4                                    ; ReturnBufferSize (4 байта)
  PUSH @StartDate                           ; Возвращаемое значение
  PUSH @ControlStartDateAttributePath       ; Имя
  PUSH 0                                    ; Родитель = корневой контейнер
  PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
  CALL

  ; получить конечную дату
  PUSH 4                                    ; ReturnBufferSize (4 байта)
  PUSH @EndDate                             ; Возвращаемое значение
  PUSH @ControlEndDateAttributePath         ; Имя
  PUSH 0                                    ; Родитель = корневой контейнер
  PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
  CALL

  ; успех
  PUSH 0
  PUSH SUCCESS
  STOP

Control.Actions.Play.Perform:
Control.Actions.Play.Check:
  ; проверить, что конечный узел доступен
  PUSH @TargetNodeId
  PUSH @IsNodeReachableFunctionNumber
  PEEK
  CALL
  BRN Play_Fail

  ; поместить текущее время в стек
  PUSH @GetTimeStampFunctionNumber
  PEEK
  CALL

  ; проверить, что дата раньше конечной даты
  DUP    ; текущее время
  PUSH @EndDate
  PEEK
  SWAP
  CMP
  BRN Play_Fail

  ; проверить, что дата позже начальной даты
  ; текущее время в стеке
  PUSH @StartDate
  PEEK
  CMP
  BRN Play_Fail

  ; успех
  PUSH 0
  PUSH SUCCESS
    STOP

  Play_Fail:
    PUSH 0
    PUSH FAILURE
    STOP

  .export Global.OnLoad
  .export Control.Actions.Play.Init
  .export Control.Actions.Play.Check
  .export Control.Actions.Play.Perform
```

Дополнительный пример программы управления включен в приложение E.

3. Потребление контента и приложения упаковки.

Ниже приведено более подробное описание иллюстративных вариантов осуществления приложения, которое потребляет DRM-защищенный контент (например, медиа-проигрывателя, текстового редактора, почтового клиента и т.д., например приложения 303a, 303c и 303d на фиг. 3), и приложения упаковки, например, приложения 303b, которое упаковывает контент, адресованный потребляющим приложениям.

3.1. Архитектура приложения, потребляющего контент.

Приложение, потребляющее контент, обычно нацелено на доступ к защищенному контенту или может составлять часть приложения общего назначения, которое также осуществляет другие функции, например, упаковки контента. В различных вариантах осуществления, приложение, потребляющее контент, может осуществлять некоторые или все из следующих функций:

обеспечивать интерфейс, посредством которого пользователь может запрашивать доступ к защищенным объектам «контент» и принимать информацию о контенте или информацию ошибки;

управлять взаимодействием с файловой системой;

распознавать формат защищенных объектов «контент»;

запрашивать механизм DRM для оценивания лицензии на фрагменты контента, для определения, можно ли дать разрешение на доступ к контенту;

проверять цифровые подписи и выполнять другие криптографические функции общего назначения, в осуществлении которых нуждается механизм DRM;

запрашивать механизм DRM для обеспечения ключей, необходимых для дешифрования защищенного контента; и/или

дешифровать защищенный контент и взаимодействовать со службами медиа-представления для представления контента.

В одном варианте осуществления, механизм клиента DRM оценивает лицензии, связанные с контентом, подтверждает или отклоняет разрешение использовать контент и передает ключи дешифрования приложению, потребляющему контент. Механизм клиента DRM также может выдавать одно или несколько обязательств и/или обратных вызовов приложению, потребляющему контент, требующих от приложения осуществлять определенные действия вследствие получения доступа к контенту.

На фиг. 13 показаны элементы, составляющие клиентское приложение 1300, потребляющее контент, в одном варианте осуществления. Согласно фиг. 13 приложение хоста 1302 является логическим центральным пунктом клиента. Он отвечает за перенос шаблона взаимодействия между другими модулями, а также взаимодействие с пользователем через пользовательский интерфейс 1304. Приложение хоста 1302 предоставляет набор услуг механизму DRM 1306 через интерфейс 1308 услуг хоста. Интерфейс 1308 услуг хоста позволяет механизму DRM 1306 получать доступ к данным, администрируемым приложением хоста 1302, а также определенным библиотечным функциям, реализованным приложением хоста 1302. В одном варианте осуществления, интерфейс 1308 услуг хоста является единственным внешним интерфейсом для механизма DRM 1306.

В одном варианте осуществления, механизм DRM 1306 не взаимодействует напрямую с мультимедийным контентом, администрируемым приложением хоста 1302. Приложение хоста 1302 логически взаимодействует с услугами контента 1310 для осуществления доступа к мультимедийному контенту, и передает механизму DRM 1306 только фрагменты данных, которые должен обрабатывать механизм. Другие взаимодействия с контентом осуществляются механизмом медиа-представления 1312. Например, в одном варианте осуществления услуги контента 1310 отвечают за получение контента от медиа-серверов и за сохранение и администрирование контента в постоянном хранилище клиента, тогда как механизм медиа-представления 1312 является подсистемой, отвечающей за доступ к мультимедийному контенту и его представление (например, на видео- и/или аудио-выходе). В одном варианте осуществления, механизм медиа-представления 1312 принимает некоторую информацию от механизма DRM 1306 (например, ключи дешифрования контента), но, в одном варианте осуществления, механизм DRM 1306 взаимодействует с механизмом медиа-представления 1312 не напрямую, но через приложение хоста 1302.

Некоторая информация, необходимая механизму DRM 1306, может быть доступна совместно с мультимедийным контентом, и ее можно получать и иметь возможность манипулировать ею посредством услуг контента 1310, однако может возникнуть необходимость получать часть этой информации посредством других услуг, например, услуги персонализации или услуги принадлежности (не показаны).

Согласно варианту осуществления, показанному на фиг. 13, криптографические операции (например, шифрование, проверка подписи, и т.д.) осуществляются блоком криптографических услуг 1314. В одном варианте осуществления, механизм DRM 1306 не взаимодействует напрямую с блоком криптографических услуг 1314, но взаимодействует косвенно, через хост 1302 (с использованием интерфейса 1308 услуг хоста), который передает его запросы. Криптографическими услугами 1314 также может пользоваться, например, механизм медиа-представления 1312 для осуществления дешифрования контента.

Очевидно, что фиг. 13 носит иллюстративный характер, и что в других вариантах осуществления различные компоненты, показанные на фиг. 13, могут быть иначе организованы, объединены, разделены, исключены, и/или могут быть добавлены новые компоненты. Например, без ограничения, логическое разделение функций между механизмом DRM и приложением хоста на фиг. 13 просто иллюстрирует один возможный вариант осуществления, и возможны различные практические реализации.

Например, механизм DRM может быть полностью или частично объединен с приложением хоста. Таким образом, очевидно, что можно использовать любое пригодное разделение функций между приложением хоста и механизмом DRM.

3.2. Архитектура упаковщика.

Ниже приведен пример функций, которые может осуществлять механизм упаковки для приложения хоста, которое упаковывает электронный контент. На практике, приложение упаковки может конкретно предназначаться для упаковки, или составлять часть приложения общего назначения, действующего на пользовательской системе, которое также осуществляет доступ к защищенному контенту (упакованному локально или в другом месте, например, в сети).

В различных вариантах осуществления, упаковывающее приложение хоста может осуществлять некоторые или все из следующих функций:

обеспечивать пользовательский интерфейс, посредством которого можно задавать информацию контента и лицензии;

шифровать контент;

создавать объекты DRM, составляющие лицензию; и/или

создавать объект «контент», который содержит или ссылается на контент и содержит или ссылается на лицензию.

На фиг. 14 показаны элементы, которые составляют приложение упаковки 1400 в одном варианте осуществления. Механизм упаковки DRM 1416 отвечает за упаковку лицензий, например, описанных здесь (например, лицензий, содержащих объекты DRM, например, объекты управления, контроллеры, протекторы и т.п.). В некоторых вариантах осуществления, механизм упаковки DRM 1416 может также связывать метаданные с лицензией для объяснения, в понятной человеку форме, что делает лицензия.

В одном варианте осуществления, приложение хоста 1402 обеспечивает пользовательский интерфейс 1404 и отвечает за получение информации, например, ссылок на контент и действие(я), которое(ые) пользователь (обычно владелец или поставщик контента) хочет осуществить (например, к кому привязать контент, какие условия пользования контентом включить в лицензию, и т.д.). Пользовательский интерфейс 1404 также может отображать информацию о процессе упаковки, например, текст выпущенной лицензии, и, в случае сбоя, причину сбоя. В некоторых вариантах осуществления, некоторая информация, необходимая приложению хоста 1402, может потребовать использование других услуг, например, услуг аутентификации или авторизации и/или принадлежности через точку доступа к службе (SAP). Таким образом, в некоторых вариантах осуществления, приложение упаковки 1400 и/или приложение хоста 1402 может нуждаться в реализации некоторых или всех из следующих служб:

Услуги медиа-формата 1406. В одном варианте осуществления, этот элемент отвечает за управление операциями медиа-формата, например, перекодирование и упаковку. Он также отвечает за шифрование контента, которое достигается посредством модуля 1408 услуг шифрования контента.

Криптографические услуги общего назначения 1410. В одном варианте осуществления, этот элемент отвечает за выдачу/проверку подписей, а также шифрование/дешифрование некоторых данных. Запросы на такие операции может выдавать точка доступа к службе 1414 или механизм упаковки DRM 1416 через интерфейс услуг хоста 1412.

Услуги шифрования контента 1408. В одном варианте осуществления, этот модуль логически отделен от криптографических услуг общего назначения 1410, поскольку он не знает о приложении. Он запускается модулем услуг медиа-формата в момент упаковки контента, с набором ключей, ранее выданных механизмом упаковки DRM 1416.

4. Вывод ключа.

Ниже описана система вывода ключа, которая естественным образом согласуется с описанными здесь предпочтительными вариантами осуществления механизма DRM и архитектурой системы, и/или которую можно использовать в других контекстах. Некоторые примеры в нижеследующем разделе взяты из иллюстративной реализации предпочтительного варианта осуществления этой системы вывода ключа, известной под названием "Scuba".

Дополнительные варианты осуществления описаны в заявке '551.

Согласно фиг. 15, в некоторых вариантах осуществления объекты «связь» 1530a, 1530b используются для распространения ключей, помимо своей основной цели - установления соотношений между узлами 1500a, 1500b, 1500c. Как описано выше, объект управления может содержать программу управления, которую можно использовать для принятия решения, следует ли удовлетворить запрос на осуществление действия. Для этого, программа управления может проверять, доступен ли конкретный узел через цепь связей. Описанные здесь методы вывода ключа пользуются наличием этой цепи связей для облегчения распространения ключа, что позволяет сделать ключ доступным механизму DRM, который

выполняет программу управления.

В одном иллюстративном варианте осуществления, каждый объект «узел» 1500a, 1500b, 1500c в данной конфигурации, которая использует необязательную систему распространения ключей, имеет набор ключей, которые используются для шифрования ключей контента и ключей других узлов. Объекты «связь» 1530a, 1530b, созданные для использования в той же конфигурации, содержат некоторые криптографические данные в качестве полезной нагрузки, что позволяет выводить информацию ключей, когда цепи связей обрабатываются механизмом DRM.

Благодаря узлам и связям, несущим ключи подобным образом, причем цепь связей 1530a, 1530b идет от узла A 1500a к узлу C 1500C, сущность (например, механизм DRM клиентского приложение хоста), которая имеет доступ к секретным ключам совместного пользования узла A 1515a, 1525a, также имеет доступ к секретным ключам совместного пользования узла C 1515c, 1525c. Возможность доступа к секретным ключам совместного пользования узла C дает сущности доступ к любому ключу контента, зашифрованному этими ключами.

4.1. Узлы, сущности и ключи.

4.1.1. Сущности.

В одном варианте осуществления системы DRM, узлы представляют собой объекты данных, не принимающие активного участия в системе. Активные участники, в этом контексте, называются сущностями. Примерами сущностей являются медиа-проигрыватели, устройства, служба подписки, упаковщики контента и т.п. С сущностями обычно связаны узлы. Сущность, которая потребляет контент, использует механизм DRM и управляет, по меньшей мере, одним объектом «узел», который составляет ее индивидуальность. В одном варианте осуществления, предполагается, что сущность имеет доступ ко всем данным объектов «узел», которым она управляет, в том числе ко всей личной информации этих объектов.

4.1.2. Узлы.

Объекты «узел», которые участвуют в иллюстративном варианте осуществления системы вывода ключа, содержат ключи как часть своих данных. В одном варианте осуществления, узлы может содержать два общих типа ключей: ключи совместного пользования и ключ конфиденциальности. В нижеследующих разделах перечислены разные типы ключей, которые можно использовать в различных вариантах осуществления. Однако очевидно, что конкретная конфигурация может использовать только подмножество этих ключей. Например, система может быть настроена на работу только с парами ключей, без использования секретных симметричных ключей. В другом случае, система может быть сконфигурирована без предоставления узлам ключей конфиденциальности, если необходимо использовать только ключи совместного пользования.

4.1.2.1. Ключи совместного пользования.

Ключи совместного пользования представляют собой пару открытого/секретного ключей и/или симметричных ключей, которые совместно используются узлом N и всеми узлами Px, для которых существует связь от Px к N, которая содержит расширения вывода ключа.

Открытый ключ совместного пользования: Kpub-share[N] Это открытая часть пары открытого/секретного ключей для шифра открытого ключа. Этот ключ обычно приходит с сертификатом, что позволяет проверять мандат сущностям, которые хотят криптографически связать с ним конфиденциальную информацию.

Секретный ключ совместного пользования: Kpriv-share[N] Это секретная часть пары открытого/секретного ключей. Сущность, управляющая узлом, призвана гарантировать, что этот секретный ключ держится в секрете. По этой причине, этот секретный ключ обычно хранится и переносится отдельно от остальной информации узла. Далее этот секретный ключ можно сделать совместно используемым с другими узлами посредством расширений вывода ключа в связях.

Симметричный ключ совместного пользования: Ks-share[N] Это ключ, который используется с симметричным шифром. Как и секретный ключ, этот ключ является конфиденциальным, и сущность, управляющая узлом, отвечает за сохранение его в секрете. Далее этот секретный ключ можно сделать совместно используемым с другими узлами посредством расширений вывода ключа в связях.

4.1.2.2. Ключи конфиденциальности.

Ключи конфиденциальности это пары ключей и/или симметричные ключи, известные только сущности, управляющей узлом, которой они принадлежат. Разница между этими ключами и вышеописанными ключами совместного пользования в том, что они не делаются совместно используемыми с другими узлами посредством расширений вывода ключа в связях.

Открытый ключ конфиденциальности. Kpub-conf[N] Это открытая часть пары открытого/секретного ключей для шифра открытого ключа. Этот ключ обычно приходит с сертификатом, что позволяет проверять мандат сущностям, которые хотят криптографически связать с ним конфиденциальную информацию.

Секретный ключ конфиденциальности. Kpriv-conf[N] Это секретная часть пары открытого/секретного ключей. Сущность, управляющая узлом, призвана гарантировать, что этот секретный ключ держится в секрете. По этой причине, этот секретный ключ обычно хранится и переносится отдельно от

остальной информации узла.

Симметричный ключ конфиденциальности: Ks-conf[N] Это ключ, который используется с симметричным шифром. Как и секретный ключ конфиденциальности, этот ключ хранится в секрете.

4.2. Криптографические элементы.

Предпочтительные варианты осуществления описанных здесь систем вывода и распространения ключа можно реализовать с использованием разнообразных криптографических алгоритмов, и они не ограничиваются каким-либо конкретным выбором криптографического алгоритма. Тем не менее, для данной(го) конфигурации или профиля, все участвующие сущности, в общем случае, должны согласовываться с набором поддерживаемых алгоритмов (термин «профиль», в целом, относится к спецификации набора фактических технологий, используемых в конкретной реализации (например, RSA для вывода ключа; XML для кодирования объектов; MP4 для формата файла и т.д.) и/или другому представлению семантического контекста, которое существует, когда объекты заданы в практической конфигурации).

В одном варианте осуществления, конфигурации включают в себя поддержку по меньшей мере одного шифра открытого ключа (например, RSA) и одного шифра симметричного ключа (например, AES).

При описании криптографических функций будет использоваться следующая система обозначений:

Ep(Kpub[N], M) означает "сообщение M, зашифрованное открытым ключом Kpub узла N с использованием шифра открытого ключа";

Dp(Kpriv[N], M) означает "сообщение M, дешифрованное секретным ключом Kpriv узла N с использованием шифра открытого ключа";

Es(Ks[N], M) означает "сообщение M, зашифрованное симметричным ключом Ks узла N с использованием шифра симметричного ключа";

Ds(Ks[N], M) означает "сообщение M, дешифрованное симметричным ключом Ks узла N с использованием шифра симметричного ключа".

4.3. Нацеливание ключей контента.

В предпочтительном варианте осуществления используется два типа криптографического нацеливания. Нацеливание ключа контента на ключи совместного пользования конечного узла означает делание этого ключа доступным для всех сущностей, которые совместно используют секретные ключи совместного пользования этого конечного узла. Нацеливание ключа контента на ключи конфиденциальности уза означает делание этого ключа доступным только для сущности, которая управляет этим узлом. Нацеливание ключа контента осуществляется путем шифрования ключа контента CK, переносимого в объекте ContentKey, с использованием одного или обоих из следующих методов.

Открытое связывание: создание объекта ContentKey, который содержит Ep(Kpub[N], CK).

Симметричное связывание: создание объекта ContentKey, который содержит Es(Ks[N], CK).

В предпочтительном варианте осуществления по возможности используется симметричное связывание, поскольку для него требуется менее вычислительно-интенсивный алгоритм, и поэтому оно менее обременительно для принимающей сущности. Однако сущность (обычно, упаковщик контента), которая создает объект ContentKey, не всегда имеет доступ к Ks[N]. Если упаковщик не имеет Ks[N], он может использовать открытое связывание, поскольку Kpub[N] не является конфиденциальной информацией, и потому его можно сделать доступным для сущностей, которым нужно произвести открытое связывание. Kpub[N] обычно делается доступным для сущностей, которым нужно нацеливать ключи контента совместно с сертификатом, который сущность может проверять для принятия решения, действительно ли Kpub[N] является ключом узла, которому можно доверить манипулировать ключом контента в соответствии с некоторой согласованной политикой (например, что узел соответствует сущности, выполняющей механизм DRM и приложение хоста, которые согласуются с функциональными, операционными политиками и политиками безопасности системы).

4.4. Вывод ключей с использованием связей.

Чтобы сущность могла иметь доступ к ключам совместного пользования всех узлов, доступных из своего узла индивидуальности, в одном варианте осуществления, объекты «связь» содержат необязательную полезную нагрузку расширения ключа. Эта полезная нагрузка расширения ключа позволяет сущностям, которые имеют доступ к открытому/секретному ключам начального узла связи, также иметь доступ к личным/секретным ключам совместного пользования конечного узла связи. Таким образом, сущность может дешифровать любой ключ контента, нацеленный на узел, который доступен из ее узла индивидуальности (если нацеливание произведено с использованием ключей совместного пользования конечного узла).

В одном варианте осуществления, когда механизм DRM обрабатывает объекты «связь», он обрабатывает полезную нагрузку расширения ключа каждой связи для обновления внутренней цепи ключей, к которой он имеет доступ. В одном варианте осуществления, полезная нагрузка расширения ключа связи L от узла F к узлу T содержит либо:

открытую информацию вывода: Ep(Kpub-share[F], {Ks-share[T],Kpriv-share[T]}), либо

симметричную информацию вывода: Es(Ks-share[F], {Ks-share[T],Kpriv-share[T]}),

где {Ks-share[T], Kpriv-share[T]} - структура данных, содержащая Ks-share[T] и Kpriv-share[T].

Открытая информация вывода используется для переноса секретных ключей совместного пользо-

вания узла T, Ks-share[T] и Kpriv-share[T] на любую сущность, которая имеет доступ к личному ключу совместного пользования узла F, Kpriv-share[F].

Симметричная информация вывода используется для переноса секретных ключей совместного пользования узла T, Ks-share[T] и Kpriv-share[T], на любую сущность, которая имеет доступ к симметричному ключу совместного пользования узла F, Ks-share [F].

Как и при нацеливании ключей контента на узлы, предпочтительная полезная нагрузка, подлежащая включению в связь, является симметричной информацией вывода. Это возможно, когда создатель связей имеет доступ к Ks-share[F]. Если нет, создатель связей делает шаг назад, включая в себя открытую информацию вывода в качестве полезной нагрузки для связи.

Предполагая, что механизм DRM, обрабатывающий связь, уже имеет Ks-share[F] и Kpriv-share[F] в своей внутренней цепи ключей после обработки связи, L[F→T], он также будет иметь Ks-share[T] и Kpriv-share[T].

Поскольку, в одном варианте осуществления, связи можно обрабатывать в любом порядке, механизм DRM может не иметь возможности производить вычисления для вывода ключа во время обработки данной связи L. Это может быть следствием того факта, что, в это время, цепь ключей механизма DRM может еще не содержать ключи начального узла этой связи. В этом случае, связь запоминается и обрабатывается снова, когда новая информация становится доступной механизму DRM, например, после обработки новой связи P. Если конечный узел связи P совпадает с начальным узлом связи L, и начальный узел связи P является доступным узлом, то начальный узел связи L также будет доступным, и на этапе вывода ключа личные ключи совместного пользования начального узла связи L добавляются в цепь ключей.

5. Примеры реализации.

Ниже приведено несколько примеров, иллюстрирующих, как различные варианты осуществления описанных здесь систем и способов можно применять на практике. Описанные здесь системы и способы могут обеспечивать широкий круг функций управления правами и других функций, откуда следует, что приведенные здесь конкретные примеры не претендуют на то, чтобы быть исчерпывающими, но просто иллюстрируют объем изобретения.

5.1. Пример. Пользователи, ПК и устройства.

Предположим, что Вы хотите реализовать систему DRM, которая связывает право на воспроизведение контента с конкретным пользователем, и Вы хотите облегчить для пользователя воспроизведение контента на всех устройствах воспроизведения, которыми он обладает. Предположим, что Вы решили снабдить пользователей программным обеспечением, которое позволяет им добавлять необходимые устройства воспроизведения (например, мобильные проигрыватели). Однако предположим также, что Вы хотите задать некоторую политику, ограничивающую количество устройств общего назначения, на которые пользователь сможет переносить контент, чтобы пользователь не имел возможности действовать как распространитель.

На основании этих системных требований, может иметь смысл, например, связывать создаваемые Вами лицензии с пользователями и устанавливать соотношения между пользователями и устройствами, которые они используют. Таким образом, в этом примере, Вы сначала можете решить, какого рода узлы Вам нужны для установления необходимых Вам видов соотношений. Например, Вы можете задать следующие типы узлов:

Пользователь (например, лицо, владеющее правами на использование контента);

ПК (например, прикладная программа, выполняющийся на персональном компьютере, которая может воспроизводить контент и указывать дополнительные устройства воспроизведения);

Устройство (например, портативное устройство для представления контента).

Каждый объект «узел» может включать в себя атрибут type (тип), который указывает, представляет ли объект пользователя, ПК или устройство.

Пусть, например, Вы решили ограничить максимальное количество объектов «узел ПК», которые могут быть присоединены к любому пользователю в конкретный момент времени, четырьмя (4). Вы решили, что не нужно ограничивать количество устройств, присоединенных к пользователю, пока Вы обеспечиваете ограничение по количеству ПК. На основании этого, можно таким образом настроить программу управления, чтобы она разрешала доступ, если можно установить соотношение между узлом «пользователь» и узлом, запрашивающим доступ. Тогда этот узел может быть либо ПК, либо устройство.

На фиг. 16 показана система, призванная удовлетворять вышеизложенным требованиям. Сервер 1600 присваивает объект «узел «пользователь»» 1602a, 1602b каждому новому пользователю 1604a, 1604b, и регулирует способность пользователей 1604a, 1604b связывать с собой устройства 1606, 1608 и ПК 1610, 1612 с целью доступа к защищенному контенту. Когда пользователь 1604a желает связать новое устройство 1606 со своим узлом «пользователь» 1602a, сервер 1600 определяет, содержит ли уже устройство 1606 информацию персонализации 1614, что может иметь место, если устройство 1606 было персонализовано во время изготовления. Если устройство не содержит информацию персонализации 1614, сервер 1600 использует эту информацию персонализации 1614 для создания связи 1616 от устройства 1606 к узлу 1602a пользователя, и передает связь 1616 устройству 1606 пользователя. Когда пользо-

ватель 1604a получает защищенный контент 1618 (например, с сервера 1600 или от какого-либо другого поставщика контента), этот контент 1618 нацеливается на узел 1602a пользователя (например, путем шифрования ключа дешифрования контента одним из секретных ключей совместного пользования, связанных с узлом 1602a пользователя), и с ним связывается лицензия 1619, указывающая условия, при которых можно осуществлять доступ к контенту. Когда пользователь 1604a пытается воспроизвести контент 1618 на устройстве 1606, механизм DRM 1620, выполняющийся на устройстве 1606, оценивает лицензию 1619, которая указывает, что контент 1618 можно воспроизводить, пока доступен узел «пользователь» 1602a. Механизм DRM 1620 оценивает связь 1616, которая показывает, что узел «пользователь» 1602a доступен из устройства 1606, и удовлетворяет запрос пользователя 1604a на доступ к контенту 1618, например, авторизуя дешифрование ключа дешифрования контента, содержащегося в лицензии 1619.

Поскольку ключ дешифрования контента, в этом примере, зашифрован с использованием секретного ключа, связанного с узлом 1602a пользователя, этот секретный ключ нужно получить, чтобы дешифровать ключ дешифрования контента. Если используются описанные здесь в другом месте необязательные методы вывода ключа, ключ узла «пользователь» можно получить, просто расшифровав информацию вывода ключа, содержащуюся в связи 1616, с использованием одного из секретных ключей устройства 1606. Дешифрованная информация вывода ключа будет содержать ключ, необходимый для дешифрования ключа дешифрования контента, содержащегося в лицензии 1619 (или информации, из которой его можно вывести или получить).

Согласно, опять же, фиг. 16, предположим, что пользователь 1604a желает связать новый ПК 1610 со своим узлом «пользователь» 1602a. Сервер 1600 проверяет, не связано ли уже с узлом «пользователь» 1602a максимальное количество ПК, и авторизует связывание ПК 1610 с узлом «пользователь» 1602a. Однако для осуществления связывания сервер 1600 должен получить информацию персонализации от ПК 1610 (например, криптографические ключи, уникальный идентификатор и т.д.). Если же прежде не был персонализован PC 1610 (что может иметь место, если пользователь просто загрузил копию программного обеспечения ПК), сервер 1600 осуществляет процесс персонализации (например, создавая объект «узел PC» с использованием протокола загрузки, описанного здесь в другом месте) или направляет пользователя к поставщику услуг, который может осуществить процесс персонализации. По завершении процесса персонализации, сервер 1600 может создать связь 1624 от ПК 1610 к узлу «пользователь» 1602a и передать связь на ПК 1610, который может продолжать пользоваться ею, пока она остается действительной.

Позже пользователь может запросить добавление дополнительных ПК, и сервер будет применять политику, которая ограничивает количество объектов «узел PC» на одного пользователя числом 4 (обычно он также предоставляет пользователям возможность, при необходимости, удалять PCs из своего активного списка).

В порядке еще одного примера, предположим, что поставщик услуг решил, что пользователи должны иметь возможность воспроизводить любой контент, которым они обладают, на любом принадлежащем им устройстве. Поставщик услуг также может пожелать позволить программному обеспечению ПК пользователя создавать связи с каждым из его устройств, не требуя от пользователя связаться с сервером 1600. В этом варианте осуществления, когда пользователь желает воспроизвести контент на новом устройстве, программное обеспечение ПК пользователя будет обращаться к конфиденциальной информации персонализации нового устройства и использовать ее для создания новой связи для этого устройства (например, связи от нового устройства к узлу 1602a пользователя). Если устройство не персонализовано, то программное обеспечение ПК может обратиться к удаленной службе или предписать устройству обратиться к удаленной службе, для осуществления процесса персонализации. Затем программное обеспечение ПК передает связь на новое устройство, и при этом новое устройство получает возможность воспроизводить контент, пока она остается действительной, поскольку, в одном варианте осуществления, если объект «связь» существует, нет необходимости создавать другую, пока не истечет срок действия объекта «связь» или он станет недействительным по другой причине.

В вышеприведенных примерах, контент нацеливается на пользователя. Для этого, приложение упаковщика выбирает новый ID для контента или использует существующий, создает ключ шифрования и соответствующий объект ContentKey, а также объект «протектор» для связывания объекта «контент» и объекта ContentKey. Затем упаковщик создает объект управления с программой управления (например, компилированной в байт-код, который может выполняться на виртуальной машине механизма DRM), что позволяет выполнять действие "воспроизведение", если и только если узел «пользователь» доступен от узла ПК или «устройство», который запрашивает действие. Обычно объекты «управление», «контроллер», «протектор» и ContentKeys по мере возможности внедряются в упакованный контент, благодаря чему ПК и устройствам не приходится получать их отдельно.

В одном варианте осуществления, когда устройство или ПК хочет воспроизвести контент, оно выполняет процесс, например, описанный выше в связи с фиг. 9. Таким образом, механизм DRM находит объект «протектор» для content ID контента, затем объект ContentKey, указанный этим протектором, затем объект «контроллер», на который ссылается этот объект ContentKey, и, наконец, объект управления,

указанный этим контроллером. Механизм DRM выполняет программу управления объекта управления, которая проверяет, доступен ли узел «пользователь». Если узел «устройство» или ПК имеет необходимые объекты «связь» для проверки наличия пути между своим узлом и узлом «пользователь», то условие выполняется, и программа управления позволяет использовать ключ, представленный в объекте ContentKey. Затем механизм медиа-представления устройства или ПК может дешифровать и воспроизвести контент.

5.2. Пример. Временный вход.

На фиг. 17 показан другой пример возможного применения описанных здесь систем и способов DRM. Этот пример аналогичен примеру, приведенному в предыдущем разделе, за исключением того, что здесь политика, которая регламентирует создание объектов «связь» между объектами «узел PC» и объектами «узел «пользователь»» допускает временный вход не более чем на 12 ч, при условии, что пользователь уже не имеет временный вход на другом ПК. Эта особенность позволяет пользователю 1700 перенести свой контент 1702 на ПК друга 1704, зайти на этом ПК 1704 на период времени и воспроизвести контент 1702 на ПК друга 1704.

Для этого создается объект «связь» 1710 с ограниченным периодом действия. В одном варианте осуществления это можно делать следующим образом.

Для простоты объяснения, предположим, что потребляющее программное обеспечение 1714 с разрешенным DRM, необходимое для воспроизведения DRM-защищенного контента 1702 уже присутствует на ПК друга 1704. Файл, содержащий контент 1702 и лицензию 1708, переносится на ПК друга 1704. Когда пользователь пытается воспроизвести контент 1702, программное обеспечение 1714 распознает отсутствие действительных объектов «связь», связывающих локальный узел ПК с узлом пользователя, который владеет контентом. Программное обеспечение 1714 предлагает пользователю представить мандат 1712 (это может обеспечиваться через имя пользователя/пароль, протокол аутентификации мобильного телефона, смарт-карту или любую систему аутентификации, разрешенную согласно политике системы) и связывается с вычислительной системой базы данных 1706. Вычислительная система базы данных 1706 проверяет атрибуты объекта «узел «пользователь»» и объекта «узел ПК», для которых запрошена связь, и проверяет, нет ли активного объекта «связь» для временного входа, который все еще действителен. При выполнении этих условий, служба базы данных 1706 создает объект «связь» 1710, связывающий объект «узел PC» друга и узел пользователя, с периодом действия, ограниченным запрошенной длительностью входа (например, менее 12 ч, для согласования с политикой в этом примере). Наличие объекта «связь» 1710 позволяет ПК друга 1704 воспроизводить контент 1702 пользователя, пока не истечет срок действия связи 1710.

5.3. Пример: управление контентом предприятия.

На фиг. 18 показана высокоуровневая архитектура иллюстративной системы 1800 для управления документацией предприятия (например, электронной почтой, документами текстового редактора, слайдами презентации, текстами мгновенного обмена сообщениями и/или т.п.). В примере, показанном на фиг. 18, приложение редактирования документов (например, текстовый редактор) 1802, почтовый клиент 1804 и сервер директорий (например, сервер активной директории) 1806 используют плагин 1808 управления цифровыми правами (DRM), уровень согласования сетевых услуг 1810, службу регистрации 1812 и службу политик 1816 для упрощения обработки документов, сообщений электронной почты, и/или т.п. в соответствии с политиками. В предпочтительном варианте осуществления, плагин DRM 1808, уровень согласования сетевых услуг 1810, служба политик 1816 и служба регистрации 1812 реализованы с использованием механизма DRM и технологии согласования услуг, описанной здесь в другом месте и в заявке '551. Например, в одном варианте осуществления, плагин DRM 1808 может содержать вышеописанный вариант осуществления механизма DRM. Очевидно, что, хотя на фиг. 18 показан вариант осуществления, в котором существующие приложения, например, текстовый редактор 1802 и почтовый клиент 1804 объединены с механизмом DRM посредством плагина, который приложения могут вызывать, в других вариантах осуществления механизм DRM может составлять неотъемлемую часть одного или обоих приложений. Также очевидно, что иллюстративную систему, показанную на фиг. 18, можно реализовать в пределах одного предприятия или можно распространить на несколько предприятий.

В иллюстрации, показанной на фиг. 18, сервер директорий 1806 может, например, содержать профили пользователей и определения групп. Например, системный администратор компании может задать группу под названием "Команда особых проектов" путем идентификации членов Команды особых проектов компании.

В одном варианте осуществления сервер директорий 1806 может содержать Сервер активной директории, выполняющий веб-услуги, например, описанные в заявке '551 (и реализованные, например, посредством стандартных технологий на основе IIS на платформе Windows®), которые выдают узлы, связи и лицензии членам группы Команда особых проектов на основании контента, к которому осуществляется доступ. Если состав членов группы меняется, выдаются новые жетоны. Для отмены прав, сервер директорий 1806 может запустить услугу метаданных безопасности, основанную на технологии, например, описанной в заявке '551 (иногда именуемой здесь технологией "NEMO"). В некоторых вариантах осуществления, может потребоваться, чтобы клиент имел значение времени на данное число или обозна-

чение времени (на основании того, насколько свежее значение задается по выбору компании (например, 1 неделя, 1 день, 1 ч, каждые 5 мин и т.д.)) для использования лицензий DRM. Например, жетон, предоставляемый услугой метаданных безопасности, может включать в себя доверенное и аутентифицируемое значение времени. В некоторых вариантах осуществления, клиент может идентифицировать ID узлов «пользователь», взаимодействуя с услугой метаданных безопасности. Метаданные безопасности можно оценивать непосредственно в контексте объектов управления лицензии для определения, все ли еще пользователь имеет данную принадлежность. Метаданные безопасности также могут возвращать агенты, которые могут определять, действительны ли соотношения, например, членство в Команде особых проектов. Таким образом, в некоторых вариантах осуществления можно усовершенствовать существующую инфраструктуру авторизации и аутентификации компании (например, Сервер активной директории компании), просто добавив несколько четко прописанных веб-услуг.

На фиг. 19 показан пример, как систему, например, показанную на фиг. 18, можно использовать для управления доступом к документу или другим его использованием. В этом примере, конкретный служащий (Джон) может часто работать над высокосекретными стратегическими проектами, и может иметь уже установленный плагин DRM 1908 для своих приложений (например, программы текстового редактора 1902, программы электронной почты 1904, программы календаря, программы или пакета программ, которая(ый) объединяет такие программы, и/или т.п.). В какой-то момент при создании документа, Джон обращается к элементу "полномочия" выпадающего меню, которое было добавлено в инструментальную панель его приложения (действие 1913). Открывается диалоговое окно полномочий, которое связывается с Сервером активной директории 1906 его компании для доступа к директории лиц и групп, зарегистрированных в системе. Он выбирает "Команда особых проектов" из списка, и выбирает дать каждому члену команды разрешение просматривать, редактировать и печатать документ. С использованием технологии согласования услуг NEMO, описанной в заявке '551, плагин DRM 1908 связывается с расширением службы политик 1916 с разрешенным NEMO до Активной директории 1906 и запрашивает копию политики для использования с целью защиты файлов Команды особых проектов (действие 1914). Когда Джон сохраняет документ, плагин DRM автоматически шифрует файл 1912, и создает объект «лицензия», нацеленный и связанный с группой под названием "Команда особых проектов" 1910. Лицензия 1910 позволяет осуществлять доступ к файлу 1912 (например, просмотр, редактирование, печать и т.д.) со стороны любого устройства, которое может создать действительную цепь связей от своего узла «устройство» к узлу «группа» Команды особых проектов.

Джон может осуществлять доступ к документу 1912, поскольку его устройство имеет связь с узлом «пользователь» Джона, а также имеет связь от узла «пользователь» Джона к узлу «группа» "Команды особых проектов". Аналогично, если он пересылает этот документ другим людям, они могут осуществлять доступ к нему только, если они также могут создать действительную цепь связей к узлу «группа» "Команды особых проектов" (например, потребовав, чтобы узел «Команда особых проектов» был доступен устройству).

Джон может сохранить файл (уже защищенный) на своем компьютере, и затем присоединить его к сообщению электронной почты (действие 1920). Например, он может открыть старое сообщение электронной почты своему начальнику (Джорджу), присоединить файл, как он обычно это делает, и отправить сообщение. Согласно фиг. 20, Джордж также имеет плагин DRM 2000, установленный на его компьютере 2014. Когда он входит на свой компьютер 2014, плагин 2000 своевременно проверяет все группы, в которые он добавлен (действие 2006), и загружает новые, обновленные связи для всех тех, срок действия которых истек (действие 2012). Если он добавлен в "Команду особых проектов" после своего последнего входа, его плагин 2000 загружает объект «связь» 2008, который связывает его узел «пользователь» с узлом «группа» "Команды особых проектов". Эта связь 2008 указывает, что узел «пользователь» Джорджа является членом узла «группа» "Команды особых проектов". В этом примере, предположим, что объект «связь» 2008 имеет дату окончания срока действия, после которой он уже недействителен (например, 3 дня).

Согласно фиг. 21, когда Джордж пытается открыть документ (действия 2130, 2132), плагин DRM 2108 проверяет внедренную (или присоединенную) лицензию, и узнает, что узел "Команда особых проектов" должен быть доступным. Его плагин 2108 строит (и удостоверяет) цепь связей 2120, 2122 от узла «устройство» его компьютера к узлу «пользователь» Джорджа; и от узла «пользователь» Джорджа к узлу «группа» "Команды особых проектов" (действие 2134). Поскольку устройство имеет действительную цепь связей 2120, 2122, его плагин 2108 разрешает доступ к файлу.

Как описано здесь в другом месте, в некоторых вариантах осуществления связи также могут нести защищенную цепь ключей. Таким образом, в некоторых вариантах осуществления, создавая цепь связей к узлу «Команда особых проектов», плагин может удостоверять не только разрешение на доступ к контенту, но также то, что он способен дешифровать цепь ключей, которая позволяет ему дешифровать контент.

Если, например, другой сотрудник (Кэрол) случайно получает электронное письмо Джона и пытается открыть документ, ее плагин DRM извлекает лицензию, связанную с файлом, и оценивает условия лицензии. Ее ПК имеет связь к ее узлу «пользователь» Кэрол; но, поскольку она не является членом ко-

манды, не существует связи от "Кэрол" к узлу «группа» "Команды особых проектов". Поскольку "Команда особых проектов" недоступна, Кэрол не разрешено обращаться к файлу.

Если Кэрол со временем добавляется с группу "Команда особых проектов", в следующий раз, когда ее плагин DRM будет обновлять ее отношения принадлежности, он обнаружит эту новую группу и загрузит объект «связь», который связывает ее узел «пользователь» с узлом «Команда особых проектов». Теперь ее плагин имеет все необходимые связи для построения цепи от ее узла «устройство» к ее узлу «пользователь» и далее к узлу «Команда особых проектов». Теперь узел «Команда особых проектов» "доступен", и она может открывать любые документы и почтовые отправления, адресованные Команде особых проектов, даже созданные до того, как она стала членом команды.

Предположим, что месяц спустя Джордж переводится на новую роль и удаляется из группы «Команда особых проектов» в Активной директории. В следующий раз, когда Джордж осуществляет вход, его плагин не принимает новый, обновленный объект «связь», связывающий узел «пользователь» Джорджа с "Командой особых проектов". Когда, спустя несколько недель, он пытается открыть файл Джона, его плагин пытается построить цепь связей к Команде особых проектов. Его ПК все еще имеет связь к узлу «пользователь» Джорджа (ПУ Джорджа все еще принадлежит ему); но связь от "Джорджа" к "Команде особых проектов" уже недействительна. Поскольку "Команда особых проектов" недоступна, ему не разрешено обращаться к файлу.

Предположим, что компания имеет политику, которая требует регистрировать в журнале доступ ко всей конфиденциальной информации. В одном таком варианте осуществления, политика для Команды особых проектов требует, чтобы все лицензии, созданные для этой группы, также требовали сбора информации о пользовании и передачи ее, например, в центральное хранилище. Таким образом, в этом примере, при оценивании (например, выполнении) программы управления в лицензии, плагин выполняет требование регистрировать доступ в журнале и делает это. Например, важные действия можно регистрировать в локальной защищенной базе данных состояний, например, описанной здесь, и, после восстановления связности сети, соответствующий контент можно сообщать через вышеописанные услуги.

На фиг. 22 показана другая иллюстративная система 2200 для управления электронным контентом на предприятии. В примере, показанном на фиг. 22, сервер LDAP 2206 используется для управления профилями пользователей, определениями групп и назначениями ролей и содержит определение группы под названием "Команда особых проектов" и определение роли "Поверенный".

Предположим, что Джон является поверенным и желает послать электронное письмо с вложением другим членам Команды особых проектов. Когда Джон устанавливает плагин DRM 2208 для своих приложений, он также устанавливает элементы в инструментальную панель своего почтового клиента. В некоторый момент при составлении сообщения электронной почты, Джон обращается к элементу "Установить полномочия" выпадающего меню, добавленного в его инструментальную панель. Плагин DRM 2208 связывается со службой политик 2216 и отображает список корпоративных политик обмена сообщениями, из которого можно выбирать. Джон выбирает "Special Project DRM Template", и плагин DRM 2208 использует протокол NEMO для запроса и удостоверения аутентичности, целостности и конфиденциальности объекта «политика», который он принимает. Политика описывает, как следует создавать лицензии, которые используют этот шаблон, в том числе, как их следует нацеливать и связывать.

Когда Джон кликает по элементу "Отправить", плагин DRM 2208 шифрует сообщение и вложение и генерирует соответствующую(ие) лицензию(и). Лицензия требует, чтобы, для доступа к электронной почте или вложению, узел «группа» Команды особых проектов или узел «группа» "Поверенные" должен быть доступным.

Лицензии связываются с зашифрованной полезной нагрузкой сообщения и зашифрованным вложением. Затем сообщение передается в список получателей с использованием стандартных функций электронной почты. Поскольку правила лицензии и шифрование не зависят от адресации электронной почты, тот факт, что неправильный получатель электронной почты может быть ошибочно включен, не подвергает опасности содержимое электронного письма или вложения.

Поскольку такой непреднамеренный получатель не имеет действительного объекта «связь», связывающего его узел «пользователь» с Командой особых проектов, ему не разрешается обращаться к контенту, если и когда он пытается сделать это. Кроме того, поскольку его устройство не имеет необходимой цепи связей (и ключей, которые они содержат), его устройство даже не имеет возможности дешифровать контент.

Однако, если непреднамеренный получатель, в свою очередь, пересылает то же самое, неизмененное электронное письмо с использованием стандартных функций электронной почты члену Команды особых проектов, этот член будет иметь объект «связь», который связывает его узел «пользователь» с узлом «группа» "Команды особых проектов", и будет способен обращаться к содержимому электронного письма.

Предположим, что другой поверенный (Билл) в компании также получил объект «связь», который связывает его с узлом «группа» "Команды особых проектов". Билл также может просматривать файл. Если он пересылает сообщение помощнику юриста (Тренту), который не является поверенным и не связан с Командой особых проектов, Трент не будет иметь объекта «связь», который связывает его с узлом

«группа» "Команды особых проектов", и он не получит доступ к документу.

Если Трент впоследствии добавляется в группу «Команда особых проектов» в директории LDAP 2206, он получает необходимые объекты «связь» и возможность доступа к ранее переправленному сообщению электронной почты.

Если, как рассмотрено выше, компания имеет политику, указывающую, что требование отчетности должно быть включено во все лицензии, то, в одном варианте осуществления, всякий раз при выполнении программы управления в одной из этих лицензий (например, когда кто-то пытается обратиться к файлу), может инициироваться событие отчетности. Этап отчетности может дополнительно включать в себя указатель, предоставлен доступ или же отклонен - это вопрос выбора реализации. При использовании такого указателя, может поддерживаться журнал количества попыток доступа к конкретному документу и состояния или иной информации каждой из них (например, успех, неудача и т.д.).

В порядке еще одного примера, предположим, что один из членов (Стивен) Команды особых проектов переходит в другую компанию для работы над особым проектом. До его перехода в другую компанию, почтовый клиент Стивена загружает локальную копию всех писем в его ящике входящих сообщений. Защищенный отчет, присоединенный к одному из этих писем, также включает в себя внедренную (или присоединенную) лицензию. Этот объект «лицензия» включает в себя как правила доступа к контенту, так и зашифрованный ключ контента. Только "отсутствующая связь", необходимая для доступа к контенту, является необходимым объектом «связь» для достижения узла «группа» "Команды особых проектов".

Поскольку в этом примере политика компании разрешает объектам «связь» оставаться действительными в течение 3 дней, объект «связь», который связывает узел «пользователь» Стивена с узлом «Команда особых проектов», будет оставаться действительным, пока он переходит и отключается. Если он попытается обратиться к файлу в автономном режиме, узел «группа» Команды особых проектов все еще будет доступен, и ему будет разрешено обратиться к файлу.

Если же Stephen останется в автономном режиме в течение более трех дней, объект «связь», связывающий его с Командой особых проектов, утратит силу. Узел «группа» Команды особых проектов станет недоступным, и Стивену не будет разрешено обращаться к файлу.

Если Стивен, в конце концов, окажется в месте, где он сможет связаться с системой компании (например, через VPN), его плагин DRM запросит обновленные копии объектов «связь» для каждой из групп, которым он принадлежит. Поскольку он все еще является членом группы "Команда особых проектов", он получит новый объект «связь» от своего узла «пользователь» к узлу «группа» Команды особых проектов. Эта связь заменит 'старую' связь, которая утратила силу и более недействительна.

Поскольку узел "Команда особых проектов" теперь доступен с использованием этой новой, обновленной связи, Стивен опять получает возможность обращаться к защищенному отчету. Новый объект «связь» будет действителен в течение 3 дней, после чего также утратит силу.

В порядке еще одного примера, предположим, что член Команды особых проектов (Салли) желает связаться с другим членом команды посредством службы мгновенного обмена сообщениями, сохранить копию сообщений и передать ее еще одному члену команды (например, в виде вложения электронной почты, дискеты, защитной заглушки и т.п.). В этом примере, клиент службы мгновенного обмена сообщениями (и, возможно, любых других продуктов для обмена сообщениями или связи, которыми компания снабжает своих сотрудников) привязывается к плагину DRM, which, как и в предыдущих примерах, обращается к политике "Special Project DRM Template", которая определяет, как нужно нацеливать и привязывать лицензии. Когда Салли пытается сохранить свои переговоры в службе мгновенного обмена сообщениями (например, выбирая "Сохранить как..."), плагин выбирает ключ шифрования (например, произвольно) и упаковывает (шифрует) текст переговоров. Согласно политике компании, плагин DRM генерирует объект «лицензия», который нацеливается и привязывается к узлу «группа» Команды особых проектов.

Файл, содержащий защищенный дубликат IM, связывается с лицензией на доступ к содержимому дубликата. Как и в предыдущих примерах, лицензия содержит как правила, которые регламентируют доступ к контенту, так и зашифрованную копию ключа контента.

Салли может перенести этот связанный файл в сообщение электронной почты, на защитную заглушку USB, дискету и т.д. с использованием стандартных процедур 'перетаскивания', и отправить его кому-то еще. При условии, что устройство получателя может создавать действительные связи к узлу «группа» особых проектов, доступ к контенту разрешен и возможен.

Предположим, что Салли передает файл Джону, который также является членом Команды особых проектов. Если Джон имеет недавно обновленный объект «связь», который идентифицирует его как члена Команды особых проектов, он сможет обращаться к файлу. Согласно политике компании этот объект «связь» содержит дату окончания срока действия, согласно которой срок его действия истекает через три дня. Поэтому, даже если Джон остается отключенным, он все же будет иметь доступ, пока эта связь остается действительной.

Если некоторое время спустя Джон уходит из Команды особых проектов на другую работу и находит в своей сумке защитную заглушку USB от Салли и пытается открыть файл с использованием своего

настольного компьютера, объект «связь», связывающий его узел «пользователь» с Командой особых проектов утратит силу. Поскольку он больше не является членом команды, плагин DRM на его устройстве уже не может найти новые, обновленные связи. Поскольку узел «группа» "Команды особых проектов" уже недостижима с его устройства, доступ не разрешается.

Полагая, что его портативный компьютер не подключался к сети с тех пор, как он сменил работу, он также пытается открыть файл с этого устройства. Поскольку максимальное предоставленное время прошло, эта связь больше не действительна. В некоторых вариантах осуществления, каждый раз, когда он пытается обратиться к файлу, может генерироваться отчет, который ставится в очередь на отправку в центральное хранилище.

Центральное хранилище принимает многочисленные отчеты о безуспешных попытках доступа к файлу и сигнализирует менеджеру по электронной почте. Менеджер напоминает Джону, что ему больше не разрешено обращаться к конфиденциальному материалу, и просит уничтожить все файлы (несмотря на то, что система указывает, что доступ не может быть предоставлен).

В порядке еще одного примера, предположим, что государственное агентство или внешний аудитор желает провести расследование или инспекцию, как Команда особых проектов обращается с конфиденциальной информацией. Чтобы помочь расследованию, компания желает продемонстрировать контрольные записи, касающиеся доступа важной информации в связи с Особым проектом.

С этой целью компания сначала сканирует все архивы незашифрованных сообщений на предмет любых сообщений, связанных с Особым проектом. К их облегчению, они обнаруживают, что, в соответствии с политикой компании, никто из сотрудников не посылал сообщений, касающихся Особого проекта, без надлежащей DRM-защиты (например, за пределы системы).

Затем компания использует записи доступа DRM для создания контрольного журнала, где подробно указано, кто и когда имел доступ к защищенной информации.

Согласно процедуре, принятой в компании, при создании группы «Команда особых проектов», в нее по умолчанию вводится начальник правового отдела (CCO). Объект «связь» для начальника правового отдела создается и сохраняется на архивном сервере, что позволяет ему, при необходимости, просматривать содержимое всех сообщений в будущем.

В этом примере политика, определенная для Команды особых проектов, указывающая, что все лицензии, создаваемые командой, должны включать в себя обязательный отчет обо всех попытках доступа к файлу, включающий в себя дату и время, UserNode, и был ли предоставлен доступ. Эти отчеты сохраняются в журнале доступа в центральном хранилище.

CCO проверяет журналы доступа для всех случаев доступа, связанных с Командой особых проектов, до того дня, когда возникло подозрение в утечке информации или нарушении правил. CCO также производит поиск в архивах электронной почты, IM и сетевого резервирования на предмет всех переданных сообщений и системных файлов на этот день и до него. Поскольку к каждому файлу присоединена лицензия (с ключом контента), и CCO имеет необходимые объекты «связь», удовлетворяющие требованиям лицензии, ему разрешен доступ к содержимому всех сообщений, к которым был совершен доступ до указанного времени.

Журналы доступа и содержимое незашифрованных сообщений поступают в полное распоряжение агентства/аудитора в порядке расследования.

В некоторых вариантах осуществления политика для Команды особых проектов также может включать в себя необходимость задавать дату окончания срока действия всех лицензий, относящихся к Особому проекту. Например, если компания лишь по закону обязана хранить записи такого рода в течение 1 года, она может указать в политике, что лицензии утрачивают силу через год после даты выпуска. В этом случае, компания может хранить записи ровно столько времени, сколько предписывает закон. После этого даже CCO не имеет к ним доступ.

В рассмотренном выше материале иногда упоминалось "нацеливание" и "привязка". В предпочтительных вариантах осуществления, нацеливание и привязка представляют два разных, но тесно связанных между собою процесса. В предпочтительных вариантах осуществления, "привязка" это, в основном, криптографический процесс, относящийся к защите ключа, который используется для шифрования контента. Когда лицензия 'привязана' к узлу (например, узлу "Команда особых проектов"), это может означать, например, что ключ контента зашифрован открытым ключом, связанным с этим узлом. Таким образом, только устройства, имеющие доступ к секретному ключу узла, будут иметь необходимый ключ для дешифрования контента (и, в предпочтительных вариантах осуществления, единственным путем к получению доступа к секретному ключу узла является дешифровка цепи связей к этому узлу); однако, просто наличие правильного секретного ключа указывает только, что устройство имеет возможность дешифровать контент, но ему еще нужно получить на это разрешение.

В предпочтительных вариантах осуществления, разрешено ли устройству обращаться к контенту, определяет программу управления в лицензии, и, в частности, как она "нацелена".

"Нацеливание" означает добавление требования в программу управления для указания, что конкретный узел (или узлы) "доступны" для осуществления пользования контентом. В вышеприведенных примерах, программа управления обычно указывает, что конкретный узел "Команда особых проектов"

доступен потребляющему устройству.

В ряде случаев может быть желательно, чтобы лицензии были нацелены на более чем один узел, например, команда по разработке нового продукта в компании ("Компания"), которая работает со многими поставщиками, поставляющими компоненты для нового совершенно секретного продукта. Предположим, что на ранних стадиях проекта, поставщик А и поставщик В (конкуренты) имеют связи к "SecretProjectX". Поставщик А желает поделиться своими идеями со всеми членами SecretProjectX, но не хочет, чтобы, по неосторожности, они стали известны поставщику В. Поставщик А может нацеливать свои лицензии так, что: ("SecretProjectX доступен") и ("Поставщик А доступен" или "Компания доступна"). Если Компания непреднамеренно предает эту информацию гласности во всём Secret Project X (в том числе, и для поставщика В), поставщик В не получит разрешения взглянуть на нее, что ограничивает всякую опасность неразглашения для Компании и ограничивает возможность поставщика А потерять свои коммерческие секреты.

5.4. Пример. Записи системы здравоохранения.

На фиг. 23 показано применение описанных здесь систем и способов для управления записями системы здравоохранения. Предположим, что медицинские записи имеют разные уровни конфиденциальности, и что желательно предоставлять разные права доступа разным сущностям в системе (например, пациентам, врачам, страховым компаниям и т.п.). Например, может быть желательно разрешать просматривать некоторые записи только пациенту, разрешать просматривать некоторые записи только врачу пациента, разрешать просматривать некоторые записи пациенту, но только в редакции врача пациента, разрешать просматривать некоторые записи всем врачам, разрешать просматривать некоторые записи всем страховым компаниям, разрешать просматривать некоторые записи только страховой компании пациента, и/или т.п.

Согласно фиг. 23, эту экосистему здравоохранения 2300 можно смоделировать с использованием объектов DRM наподобие узлов и связей, например, описанных здесь в другом месте. Например, узлы можно присваивать пациенту 2302, врачам 2304 пациента, страховой компании 2306 пациента, устройствам (2308, 2310) пациента, конкретному одному из врачей 2312 пациента, вычислительным устройствам 2314, 2316 врача, группе всех врачей 2318, группе врачей определенной специализации 2320, медицинскому учреждению 2322, страховой компании 2324, вычислительным устройствам, используемым страховой компанией 2326, группе всех страховых компаний 2328, и т.п.

Предположим, что врач пациента использует свой ПК для создания медицинской записи, касающейся пациента. Например, медицинская запись может содержать шаблон документа содержащий поля для примечаний, диагнозов, рецептов, инструкций для пациента и/или т.п. Шаблон также может позволять врачу выбирать политики безопасности для управления документом и/или отдельным его полем. Например, приложение врача может предоставлять возможность выбора из набора стандартных политик безопасности, и, получив выбор врача, может автоматически генерировать лицензию на основании этих выборов и связывать защищенный (например, зашифрованный) контент медицинской записи.

В целях этого примера, предположим, что лицензия предоставляет доступ для просмотра пациенту, всем поставщикам услуг здравоохранения, которые лечат пациента, и всем страховым компаниям, которые обеспечивают покрытие для пациента. Кроме того, предположим, в целях иллюстрации, что лицензия предоставляет права редактирования только кардиологу в медицинском учреждении x.

Приложение упаковки принимает ввод, указывающий политику врача (который может просто содержать кликанье мышкой по стандартному шаблону) и генерирует лицензию, которая включает в себя программу управления, например, приведенную ниже:

```
                Action.Edit.Perform() {
                    if (IsNodeReachable("MedicalFoundationX") &&
                        IsNodeReachable("Cardiologist")) {
                     return new ESB(ACTION_GRANTED);
                } else {
                    return new ESB(ACTION_DENIED);
                }
            }

            Action.View.Perform() {
                if (IsNodeReachable("PatientY") ||
                    IsNodeReachable("HCPsPatientY") ||
                    IsNodeReachable("ICsPatientY") {
                    return new ESB(ACTION_GRANTED);
                } else if (EmergencyException == TRUE) {
                    return new ESB(ACTION_GRANTED, new NotificationObligation());
    }
                else {
                    return new ESB(ACTION_DENIED);
                }
            }
```

Медицинскую запись и связанную с ней лицензию можно затем сохранить в центральной базе данных медицинских записей, базе данных, которой управляет конкретное медицинское учреждение, и/или т.п. Если пациент Y впоследствии посещает другого поставщика услуг здравоохранения и авторизует этого поставщика услуг здравоохранения как одного из его доверенных поставщиков услуг здравоохранения (например, подписывая форму авторизации), этот поставщик услуг здравоохранения получает связь к approved узлу «доверенный поставщик услуг здравоохранения» пациента y, которую поставщик услуг здравоохранения будет хранить на своей компьютерной системе. Если этому поставщику услуг здравоохранения понадобится обратиться к медицинской записи, созданной врачом x, он сможет получить доступ для просмотра этой медицинской записи, поскольку узел «доверенный поставщик услуг здравоохранения» пациента y будет доступен от компьютерной системы нового поставщика услуг здравоохранения. Если же, недоверенный поставщик услуг здравоохранения захочет получить копию (зашифрованной) медицинской записи, он не сможет обратиться к ней ввиду отсутствия необходимых узлов (т.е. узла пациента y, узла для всех доверенных поставщиков услуг здравоохранения пациента y и узла для всех доверенных страховых компаний пациента y), которые были бы доступны от его вычислительной системы.

Заметим, однако, что показанная выше иллюстративная программа управления включает в себя доминирующую особенность, которую можно вызывать, например, в экстренных случаях, если, например, поставщик услуг здравоохранения нуждается в доступе к защищенной медицинской записи, но не может выполнить условия программы управления (например, потому, что поставщик услуг здравоохранения, пытающийся сделать экстренный доступ к медицинской записи, ранее не был зарегистрирован как поставщик услуг здравоохранения пациента Y). Однако заметим также, что вызов исключения экстренного доступа приводит к автоматической записи информации, касающейся вызова и/или иных обстоятельств, и, в этом примере, также приводит к отправке извещения (например, предпочтительному поставщику услуг здравоохранения пациента, т.е. сущности, в явном виде авторизованной пациентом, и/или самому пациенту). Связывание таких обязательств с экстренным исключением может препятствовать злоупотреблению исключением, поскольку будет существовать запись о злоупотреблении.

Очевидно, что эта иллюстративная программа представлена для облегчения объяснения определенных вариантов осуществления описанных здесь систем и способов. Например, включает ли в себя система поддержку экстренных исключений, обычно зависит от требований и желаний архитектора системы. Таким образом, например, некоторые варианты осуществления могут не поддерживать экстренные исключения, другие могут поддерживать экстренные исключения, но ограничивать класс сущностей, которые могут вызывать такие исключения, классом "все врачи" (например, за счет требования, чтобы флаг EmergencyException был задан равным "истина" И узел «все врачи» был доступен), и другие все же могут поддерживать экстренные исключения, но не связывать с ними принудительные обязательства (поскольку неспособность выполнить обязательство, в предпочтительном варианте осуществления, сделает контент недоступным), опираясь вместо этого на нетехнические, правовые или институциональные средства применения (например, путем оказания доверия поставщикам услуг здравоохранения, не злоупотребляющим возможностью вызывать исключение, и/или опираясь на промышленную сертификацию и правовую систему для предотвращения злоупотреблений).

Еще одно изменение, которое можно внести в вышеприведенные примеры, состоит в том, что можно потребовать более сильное доказательство того, что к медицинской записи обратился действительно врач, или врач с конкретным именем, а не кто-то другой, сидящий за компьютером, который врач использует для доступа к записям (и, таким образом, компьютером, потенциально содержащим связи, необходимые для удовлетворения анализа доступности). Такую усиленную форму аутентификации можно осуществлять любым подходящим способом. Например, ее можно полностью или частично осуществлять на уровне приложения или системы, защищая компьютер врача и/или программное обеспечение, используемое для доступа к медицинским записям, с использованием паролей, защитных заглушек, механизмов биометрической идентификации и/или т.п. Альтернативно или дополнительно, программы управления, связанные с определенными медицинскими записями сами могут включать в себя обязательство или условие, требующее такой усиленной идентификации, например, проверку наличия защитной заглушки, требование, чтобы хост получил пароль, и/или т.п.

5.5. Пример. Подписки.

На фиг. 24 показано, как представленные здесь системы и способы можно использовать в контексте службы электронной подписки. Пусть, например, пользователь (Алиса) желает получить подписку на джазовую музыку от поставщика услуг интернета (XYZ ISP). Поставщик услуг интернета может предлагать разнообразные варианты подписки, включающие в себя пробную подписку, не требующую оплаты, но которую можно использовать только для воспроизведения контента подписки пять раз до истечения срока (например, проиграть одну песню пять раз, проиграть пять разных песен по одному разу, и т.п.). Пробная подписка также может предусматривать обеспечение контента в слегка ухудшенной форме (например, с пониженным качеством звучания или разрешением). Алиса использует свой персональный компьютер для доступа к интернет-сайту поставщика услуг и выбирает пробную подписку. Тогда поставщик услуг создает объект «связь» 2400 и агент 2401 и передает их на персональный компьютер 2406

Алисы. Агент 2401 способен инициализировать состояние защищенной базы данных состояний Алисы, которая будет использоваться для отслеживания, сколько раз Алиса использовала пробный контент. Связь 2400 идет от узла учетной записи ISP Алисы (Aflnca@XYZ_ISP) 2402 к узлу подписки 2404 и включает в себя программу управления, которая, когда Алиса запрашивает воспроизведение фрагмента контента, проверяет текущее значение переменной состояния, заданной агентом 2401, чтобы определить, разрешены ли дополнительные воспроизведения.

Когда Алиса загружает фрагмент контента на свой ПК и пытается воспроизвести его, механизм DRM на ее ПК оценивает лицензию, связанную с контентом, которая указывает, что узел подписки 2404 должен быть доступным для воспроизведения контента. Алиса ранее зарегистрировала свой ПК у своего ISP, и при этом получила связь 2405 от своего узла ПК 2406 к своему узлу учетной записи 2402. Таким образом, механизм DRM обладает объектами «связь» 2405, 2400, соединяющими узел ПК 2406 с узлом подписки 2404; однако, прежде чем удовлетворить запрос Алисы на воспроизведение контента, механизм DRM сначала определяет, действительны ли связи, выполняя любые программы управления, содержащиеся в связях. При выполнении программы управления в связи 2400, механизм DRM проверяет элемент базы данных состояний для определения, осуществлено ли воспроизведение 5 раз, и, если нет, удовлетворяет запрос на воспроизведение контента, но также выдает обязательство на приложение хоста. Обязательство требует, чтобы хост ухудшил контент до представления. Приложение хоста определяет, что оно способно выполнить это обязательство, и переходит к представлению контента. Для обеспечения Алисы предварительным просмотром контента до учета этого контента в отношении ее пятикратного пробного воспроизведения, программа управления должна также включать в себя обратный вызов, который проверяет, например, через 20 с после удовлетворения запроса на воспроизведение фрагмент контента, продолжается ли воспроизведение контента. Если контент все еще воспроизводится, счетчик воспроизведения уменьшается, а в противном случае - нет. Таким образом, Алиса может выбирать любые из элементов контента, предлагаемых службой подписки, и воспроизводить любые пять из них до истечения срока пробной подписки.

По истечении срока пробной подписки Алисы, она решает приобрести полную, месячную подписку, которая позволяет ей воспроизводить столько элементов контента, сколько она желает, за месячную плату. Алиса использует свой ПК для заказа подписки и получает связь 2410 от своего узла учетной записи 2402 к узлу подписки 2404. Связь включает в себя программу управления, указывающую, что связь действительна только в течение одного месяца (например, программа управления проверяет элемент базы данных состояний, чтобы определить, истек ли месяц с момента выдачи связи). Эта связь 2410 передается на ПК Алисы совместно с программой-агентом, которая способна инициализировать соответствующий элемент базы данных состояний механизма DRM ПК, указывающий дату выдачи связи. Когда Алиса загружает фрагмент контента из службы подписки и пытается воспроизвести его, ее механизм DRM ПК определяет, что существует путь к узлу подписки, состоящий из связей 2405, 2410. Механизм DRM выполняет любые программы управления, содержащиеся в связях 2405, 2410 для определения, действительны ли связи. Если с момента выдачи связи 2410 прошло меньше месяца, программа управления в связи 2410 возвращает результат, указывающий, что связь 2410 все еще действительна, и запрос Алисы на воспроизведение фрагмента контента удовлетворяется. Если Алиса пытается воспроизвести фрагмент контента, который она ранее получила в течение своего пробного периода, механизм DRM на ее ПК осуществляет такой же анализ и удовлетворяет ее запрос. Поскольку лицензия, связанная с фрагментом контента, полученная в течение пробного периода, указывает, что если переменна TrialState в защищенной базе данных не задана, то единственное условие состоит в том, что узел подписки должен быть доступным, Алиса может снова обратиться к этому контенту, поскольку узел подписки снова доступен от ПК Алисы, на этот раз через связь 2410, а не связь 2400, которая уже недействительна. Таким образом, Алисе не нужно получать вторую копию элемента контента для замены копии, которую она получила в течение бесплатного пробного предложения. Аналогично, если Алиса получает фрагмент контента подписки от своего друга Боба, который также подписался на ту же услугу, Алиса, в этом примере, тоже сможет воспроизводить этот контент, поскольку лицензия контента просто требует, чтобы узел подписки был доступен, не обязательно через ПК или учетную запись Боба.

Очевидно, что вышеприведенные примеры призваны просто иллюстрировать некоторые функции, которые могут быть обеспечены описанными здесь системами и способами, и не призваны указывать, что подписки должны быть реализованы именно вышеописанным образом. Например, в других вариантах осуществления, лицензия, связанная с фрагментом контента подписки можно связывать с узлом пользователя, а не с узлом подписки, что препятствует двум подписчикам, например, Бобу и Алисе, совместно использовать контент, что возможно в вышеописанном примере. Очевидно, что возможны многие другие вариации вышеприведенных примеров.

В нижеприведенной таблице представлен некоторый иллюстративный псевдокод для агента, связи и программ управления лицензии в вышеописанном примере:

```
==========
```

Пробная подписка обеспечивает доступ к контенту подписки в объеме до 5 фрагментов. Контент помечается как представляемый только спустя 20 секунд представления. Контент, представляемый в пробном режиме, будет ухудшен приложением представления.

Реальная подписка будет обновляться каждый месяц и не имеет подобных ограничений по количеству и качеству представлений.

Ниже представлен код агента:

```
==========
TrialAgent() {
    SetObject("TrialState", 5);
}
==========

Код программы управления в пробной связи:
==========
Control.Link.Constraint.Check() {
    if (GetObject("TrialState", 5) > 0) {
        return SUCCESS;
    } else {
        return FAILURE;
    }
}
==========
```

Когда Алиса действительно регистрируется в службе подписки, она получает связь (начало: Алиса, конец: Подписка) и агент

Ниже представлен код агента:

```
==========
RealSubscriptionAgent() {
    // очистить TrialState, если присутствует
    trialState = GetObject("TrialState");
    if (trialState != NULL) {
        SetObject("TrialState", NULL); // очистить
    }
}
==========

Ниже представлен код связи:
==========
Control.Link.Constraint.Check() {
    if (GetTrustedTime() < ExpirationDate) {
        return SUCCESS;
    } else {
        return FAILURE;
    }
}
==========
```

Все лицензии контента, нацеленные на подписку, имеют одну и ту же программу управления:

```
==========

Action.Play.Perform() {
    // сначала проверить, доступен ли узел подписки
    if (!IsNodeReachable("SubscriptionNode")) {
        return new ESB(ACTION_DENIED);
    }

    // теперь проверить, присутствует ли TrialState
    if (GetObject("TrialState) != NULL) {
        // мы в пробном режиме: нам нужны обратный вызов и обязательство
        return new ESB(ACTION_GRANTED,
                new OnTimeElapsedCallback(20, DecrementCounter),
                new DegradeRenderingObligation());
    } else {
// мы в режиме платной подписки: просто возвратить ACTION_GRANTED
        return new ESB(ACTION_GRANTED);
    }
}

// код функции обратного вызова OnTimeElapsed
DecrementCounter() {
    SetObject("TrialState", GetObject("TrialState") - 1);
}
==========
```

Опять же, согласно фиг. 24, Алиса также имеет учетную запись 2420 у своего поставщика услуг мо-

бильной связи, которая остается действительной, пока она остается подключенной к сети.

Алисе не требуется производить специальный платеж за подписку, взамен чего она получает связь; вместо этого обновление связей 2424 автоматически поступает на ее телефон, когда она подключается к сети. Эти связи позволяют ей осуществлять доступ к любым элементам контента или услугам, предлагаемым поставщиком услуг мобильной связи, которые имеют лицензии, которые требуют только, чтобы узел подписки 2422 был доступен. Если Алиса сменит поставщика услуг мобильной связи, она не сможет осуществлять доступ к ранее полученному контенту по истечении срока действия ее связей 2424.

На фиг. 25 показан пример того, как поставщик услуг может взаимодействовать с доменом домашней сети 2500. В этом примере, устройства зарегистрированы в домене домашней сети, где применяется политика, которая позволяет до 5 устройств принадлежать домену в любой момент времени. Хотя поставщик услуг кабельного телевидения семьи Смитов не обеспечивает программное обеспечение менеджера доменов, используемое для установления домена домашней сети 2500, поставщик услуг кабельного телевидения знает, что менеджер доменов реализован сертифицированным поставщиком программного обеспечения менеджера домена домашней сети, и, таким образом, доверяет программному обеспечению менеджера доменов действовать, как положено. Согласно фиг. 25, семья Смитов подключает телефон и ПК Алисы, PVR Карла и PSP Джо к домену 2500, в результате чего создаются связи от каждого из этих устройств к узлу домена 2500. В случае приема нового контента, например, на PVR, услуги обнаружения, например, описанные в заявке '551, позволяют другим устройствам в домене автоматически получать контент и любые необходимые необходимый связи. Создаются связи от узла «домен» 2500 к узлу «учетная запись» 2502 поставщика услуг. Контент некоторых поставщиков услуг кабельного телевидения имеет лицензию с обязательством, что перемотка вперед и назад должна быть отключена, чтобы можно было показывать рекламу. PVR Карла и ПК Алисы способны выполнять обязательство и, таким образом, могут воспроизводить контент. Мобильный телефон Алисы не может выполнять обязательство и поэтому не получает доступ к контенту.

5.6. Дополнительные примеры: совместное использование контента и прав.

Как показано в предыдущих примерах, представленные здесь варианты осуществления систем и способов позволяют естественным образом делать электронный контент совместно используемым. Например, описанные здесь системы и способы можно использовать для того, чтобы потребители могли совместно использовать развлекательный контент со своими друзьями и членами семьи, и/или наслаждаться им на всех своих семейных устройствах, и, в то же время, препятствовать более широкому, неавторизованному распространению. Например, можно использовать автоматизированные услуги обнаружения и оповещения равноправных устройств, в результате чего, когда одно устройство получает контент или соответствующие права, другие устройства могут автоматически узнавать об этом контенте, тем самым обеспечивая виртуальную распределенную библиотеку, которая может автоматически обновляться. Например, в одном варианте осуществления, если один пользователь получает контент или права на портативном устройстве в одном месте, а затем приходит домой, семейные устройства пользователя могут автоматически обнаруживать и использовать эти права. Напротив, если пользователь получает права на устройстве в своей домашней сети, его портативные устройства могут обнаруживать и переносить этот контент для использования в другом месте. Предпочтительные варианты осуществления описанных здесь систем и способов можно использовать для создания объектов услуг и прав, которые позволяют полностью автоматизировать вышеописанные сценарии с использованием, например, методов обнаружения и инспекции услуг, описанных в заявке '551. Например, устройства, зарегистрированные в конкретном домене, могут предоставлять услуги друг другу (например, путем совместного использования прав и контента), и/или можно вызывать удаленные службы для облегчения локального совместного пользования контентом. Описанные системы и способы позволяют создавать структуры DRM, которые не нацелены на предотвращение создания копий само по себе, но призваны работать в гармонии с сетевой технологии, позволяя совместно использовать контент, в то же время, не позволяя потребителям становиться незаконными распространителями контента.

Предпочтительные варианты осуществления описанных здесь систем и способов DRM также позволяют определять права без обширной характеристики типов выражений прав некоторых других систем DRM. Вместо этого, предпочтительные варианты осуществления используют набор специализированных объектов прав, которые могут взаимодействовать контекстуально. Эти объекты описывают соотношения и управляющие элементы между сущностями, например пользователями, устройствами, контентом и их группами. Например, такие контекстуальные взаимодействия могут позволять устройству определять, что данный фрагмент контента можно воспроизводить, поскольку (a) контент получен от законной услуги контента, на которую пользователь в настоящее время подписан, (b) пользователь входит в конкретную семейную группу, и (c) устройство связано с этой конкретной семейной группы. Существуют многочисленные типы соотношений, например, описанные в этом примере, которые пользователи понимают интуитивно, и предпочтительные варианты осуществления описанных здесь систем и способов способны создавать системы, которые естественным образом понимают эти виды соотношений. Соотношения между сущностями можно создавать, уничтожать и изменять со временем, и предпочтительные варианты осуществления обеспечивают естественный способ определения прав в динамиче-

ской сетевой среде - среде, которую потребители могут естественным образом понимать. Тем не менее, если распространитель контента хочет использовать более традиционный подход к выражению прав, предпочтительные варианты осуществления могут охватывать и его. Например, можно использовать инструменты для перевода традиционных выражений прав в наборы объектов, например, описанных выше, и/или можно реализовать механизм DRM, который работает непосредственно на таких выражениях прав. Альтернативно, в некоторых вариантах осуществления, устройства не понимают такие традиционные выражения прав, и не подчиняются их ограничениям.

Предпочтительные варианты осуществления описанных здесь систем и способов также имеют очень общее обозначение медиа-услуг. Широковещательная служба и интернет-служба загрузки или подписки являются примерами медиа-услуг. Ограничения, связанные с этими услугами, могут затруднять совместное использование контента. Согласно предпочтительным вариантам осуществления описанных здесь систем и способов, контент можно получать в рамках широковещательных, широкополосных и мобильных услуг, и совместно использовать в группе сетевых устройств в доме, включающей в себя портативные устройства. Альтернативно или дополнительно, услуги могут предоставляться отдельными устройствами в режиме взаимодействия равноправных устройств посредством беспроводной связи. Например, новое поколение сотовых телефонов с функцией WiFi может обеспечивать услуги каталогов контента для других устройств. Такая услуга позволяет другим устройствам "видеть", что контент доступен для совместного использования от устройства. Услуга обеспечивает информацию, которую можно использовать для определения прав, что позволяет принимать или легко исключать любые ограничения.

Предпочтительные варианты осуществления описанных здесь систем и способов не ограничиваются одной услугой или одной платформой. Как объяснено выше, предпочтительные варианты осуществления способны работать с многочисленными услугами, в том числе "персональными" услугами. Это приобретает все большее значение по мере распространения домашних и персональных сетей. Например, в настоящее время доступны цифровые камеры с возможностью связи по WiFi, что создает большое удобство для распространения фотографий по сетям. Полезно иметь возможность автоматического распространения фотографий, но камере придется иметь дело со многими разными сетями по мере ее перемещения. Автоматизированное распространение удобно, но личные фотографии, конечно, являются личными. Варианты осуществления описанных здесь систем и способов позволяют легко распространять фотографии в семье на семейных устройствах, но не с любыми устройствами, которые могут встретиться камере в сети. В общем случае, чем больше устройств становятся сетевыми, тем важнее становится управлять правами на весь контент на этих устройствах. Хотя целью подключения к сети является обеспечение совместного использования информации на сетевых устройствах, сети будут перекрываться и сливаться друг с другом. Сети позволяют легко обеспечивать совместное использование контента, но его совместное использование не должно быть бесконтрольным. Таким образом, желательно иметь систему DRM, которая адаптируется к сети и которая может использовать контекст, обеспечиваемый контентом, пользователем, сетью и характеристиками устройств, для определения, следует ли обеспечивать совместное использование контента и как это нужно делать. Предпочтительные варианты осуществления описанных здесь систем и способов обеспечивают такой подход.

6. Иллюстративная архитектура для потребления и упаковки контента.

Ниже приведено описание базовой архитектуры потребляющего приложения (например, медиапроигрывателя), которое потребляет DRM-защищенный контент, и приложения упаковки (например, приложения, установленного на сервере), которое упаковывает контент, адресованный потребляющим приложениям.

6.1. Архитектура клиента.

Ниже приведен пример функций, которые механизм DRM согласно иллюстративному варианту осуществления может осуществлять для приложения хоста, которое потребляет контент.

6.1.1. Интерфейс приложения хоста к механизму DRM.

Хотя в предпочтительном варианте осуществления для механизмов DRM не требуются API, ниже приведены высокоуровневые описания разновидности интерфейса, обеспечиваемого иллюстративным механизмом DRM (именуемым здесь механизмом DRM "Octopus") к приложению хоста в одном иллюстративном варианте осуществления:

Octopus::CreateSession(hostContextObject) → Session.

Создает сеанс, заданный контекстом приложения хоста. Объект «контекст» используется механизмом DRM Octopus для производства обратных вызовов в приложение.

Session::ProcessObject(drmObject).

Эта функция должна вызываться приложением хоста, когда он сталкивается с определенными типами объектов в медиа-файлах, которые можно идентифицировать как принадлежащие подсистеме DRM. Такие объекты включают в себя программы управления контентом, жетоны принадлежности и т.д. Синтаксис и семантика этих объектов непрозрачна для приложения хоста.

Session::OpenContent(contentReference) → Content.

Приложение хоста вызывает эту функцию, когда ему нужно взаимодействовать с файлом мультимедийного контента. Механизм DRM возвращает объект Content, который затем можно использовать для извлечения информации DRM о контенте и взаимодействия с ним.

Content::GetDrmInfo().

Возвращает метаданные DRM о контенте, которые отсутствуют в обычных метаданных для файла.

Content::CreateAction(actionInfo) → Action.

Приложение хоста вызывает эту функцию, когда оно хочет взаимодействовать с объектом Content. Параметр ActionInfo указывает, какого типа действие должно осуществлять приложение (например, воспроизведение), а также, если необходимо, любые связанные с ним параметры. Функция возвращает объект Action, который затем можно использовать для осуществления действия и извлечения ключа контента.

Action::GetKeyInfo().

Возвращает информацию, которая необходима подсистеме дешифрования для дешифрования контента.

Action::Check().

Проверяет, авторизует ли подсистема DRM осуществление этого действия (т.е. будет ли выполнено Action::Perform()).

Action::Perform().

Осуществляет действие и обеспечивает любые последствия (в том числе, побочные эффекты), согласно правилу, которое управляет этим действием.

6.1.2. Интерфейс механизма DRM к услугам хоста.

Ниже приведен пример разновидности интерфейса услуг хоста, необходимый механизму DRM, согласно иллюстративному варианту осуществления, от приложения хоста согласно иллюстративному варианту осуществления.

HostContext::GetFileSystem(type) → FileSystem.

Возвращает виртуальный объект FileSystem, к которому подсистема DRM имеет эксклюзивный доступ. Этот виртуальный FileSystem можно использовать для сохранения информации состояния DRM. Данные в этом FileSystem может читать и записывать только подсистема DRM.

HostContext::GetCurrentTime().

Возвращает текущие дату/время, поддерживаемые системой хоста.

HostContext::GetIdentity().

Возвращает уникальный ID этого хоста.

HostContext::ProcessObject(dataObject).

Передает обратно услугам хоста объект данных, который может быть внедрен в объект DRM, но который подсистема DRM идентифицировала как управляемый хостом (например, сертификаты).

HostContext::VerifySignature(signatureInfo).

Проверяет действительность цифровой подписи на объекте данных. В одном варианте осуществления объект signatureInfo содержит информацию, эквивалентную информации, найденной в элементе XMLSig. Услуги хоста отвечают за управление ключами и сертификатами ключей, необходимыми для удостоверения подписи.

HostContext::CreateCipher(cipherType, keyInfo) → Cipher.

Создает объект Cipher, который подсистема DRM может использовать для шифрования и дешифрования данных. Задается минимальный набор типов шифра, и для каждого из них реализация шифра требует формат для описания информации ключа.

Cipher::Encrypt(data).

Cipher::Decrypt(data).

HostContext::CreateDigester(digesterType) → Digester.

Создает объект Digester, который подсистема DRM может использовать для вычисления защищенного хэша на некоторых данных. В одном варианте осуществления, можно задать минимальный набор типов дайджеста.

Digester::Update(data).

Digester::GetDigest().

6.1.3. Диаграмма последовательности UML.

На фиг. 26 показано использование иллюстративных API, описанных в предыдущих разделах, и взаимодействия, которые имеют место между приложением хоста и механизмом клиента DRM в иллюстративном варианте осуществления.

6.2. Иллюстративная архитектура упаковщика.

Ниже приведен пример функций, которые может осуществлять механизм упаковки для приложения хоста, которое упаковывает контент. На практике, приложение упаковки может конкретно предназначаться для упаковки, или составлять часть приложения общего назначения, действующего на пользовательской системе, которое также осуществляет доступ к защищенному контенту (упакованному локально

или в другом месте, например, в сети).

6.2.1. Интерфейс приложения хоста к механизму упаковки.

В этом разделе приведено высокоуровневое описание иллюстративного API между приложением хоста и механизмом упаковки, используемым в связи с иллюстративным механизмом DRM, именуемым "Octopus".

Octopus::CreateSession(hostContextObject) → Session.

Создает сеанс для данного контекста приложения хоста. Объект «контекст», возвращаемый этой функцией, используется механизмом упаковки для осуществления обратных вызовов в приложение.

Session::CreateContent(contentReferences[]) → Content.

Приложение хоста вызывает эту функцию для создания объекта Content, который будет связан с объектами «лицензия» на последующих этапах. Наличие более одной ссылки на контент в массиве contentReferences подразумевает, что они связаны друг с другом в пучок (например, одна дорожка аудио и одна дорожка видео), и что выданная лицензия должна быть нацелена на них как на одну неделимую группу.

Content::SetDrmInfo(drmInfo).

Параметр drmInfo указывает метаданные лицензии, которая будет выдана, drmInfo выступает в роли руководства по трансляции лицензии в байт-код для виртуальной машины.

Content::GetDRMObjects(format) → drmObjects.

Эта функция вызывается, когда приложение хоста готово получить drmObjects, созданные механизмом упаковки. Параметр format указывает предполагаемый формат этих объектов (например, XML или двоичные атомы).

Content::GetKeys() → keys[].

Эта функция вызывается приложением упаковки хоста, когда ему нужны ключи для шифрования контента. В одном варианте осуществления, существует по одному ключу для каждой ссылки на контент.

6.2.2. Интерфейс механизма упаковки к услугам хоста.

Ниже приведен пример типа интерфейса, который нужен иллюстративному механизму упаковки для обеспечения приложения хоста в одном варианте осуществления.

HostContext::GetFileSystem(type) → FileSystem.

Возвращает виртуальный объект FileSystem, к которому подсистема DRM имеет эксклюзивный доступ. Этот виртуальный FileSystem можно использовать для сохранения информации состояния DRM. Данные в этом FileSystem может читать и записывать только подсистема DRM.

HostContext::GetCurrentTime() → Time.

Возвращает текущие дату/время, поддерживаемые системой хоста.

HostContext::GetIdentity() → ID.

Возвращает уникальный ID этого хоста.

HostContext::PerformSignature(signatureInfo, data).

Некоторые объекты DRM, созданные механизмом упаковки, должны быть доверенными. Эта услуга, обеспеченная хостом, будет использоваться для подписывания указанного объекта.

HostContext::CreateCipher(cipherType, keyInfo) → Cipher.

Создает объект Cipher (объект, способный шифровать и дешифровать данные), который механизм упаковки может использовать для шифрования и дешифрования данных. В одном варианте осуществления, объект Cipher используется для шифрования данных ключа контента в объекте ContentKey.

Cipher::Encrypt(data).

Шифрует данные.

Cipher::Decrypt(data).

Дешифрует данные.

HostContext::CreateDigester(digesterType) → Digester.

Создает объект Digester, который механизм упаковки может использовать для вычисления защищенного хэша на некоторых данных.

Digester::Update(data).

Передает данные в объект Digester.

Digester::GetDigest().

Вычисляет дайджет.

HostContext::GenerateRandomNumber().

Генерирует случайное число, которое можно использовать для генерации ключа.

На фиг. 27 показана диаграмма UML, демонстрирующая пример использования вышеописанных иллюстративных API и взаимодействий, которые имеют место между приложением хоста и механизмом упаковки в одном иллюстративном варианте осуществления.

7. Объекты.

В этом разделе приведена дополнительная информация, относящаяся к объектам DRM, которые

служат строительными блоками иллюстративной реализации механизма DRM. Сначала дадим сравнительно общий обзор типов объектов, которые механизм DRM использует для защиты и администрирования контента. Затем приведем более подробное описание этих объектов и информации, которую они несут, а также некоторых иллюстративных структур данных, используемых в одном иллюстративном варианте осуществления.

7.1. Объекты DRM для защиты и администрирования контента.

Как описано выше в связи с фиг. 6, объекты управления контентом (иногда именуемые, совместно с объектами «узел» и «связь», объектами DRM) используются для связывания правил и условия пользования с защищенным контентом. Совместно, эти объекты образуют лицензию.

Согласно фиг. 6, данные, представленные объектом «контент» 614, зашифрованы с использованием ключа. Этот ключ, необходимый для дешифрования контента, представлен объектом ContentKey 602, и связь между контентом и ключом, используемым для его шифрования, представлена объектом «протектор» 604. Правила, которые регламентируют использование ключа дешифрования, представлены объектом управления 608, и связь между ContentKey 602 и объектом управления 608 представлена объектом «контроллер» 606. В одном варианте осуществления, доверенные системы используют ключ дешифрования контента только согласно правилам, выраженные байт-кодом в объекте управления 608. На фиг. 28A показана более подробная иллюстрация лицензии, например, показанной на фиг. 6, и показана схема подписи, которая используется в одном варианте осуществления.

7.1.1. Общие элементы.

В одном варианте осуществления, объекты совместно используют общие основные особенности: каждый из них может иметь ID, список атрибутов и список расширений.

7.1.1.1. ID.

Объекты, на которые ссылаются другие объекты, имеют уникальный ID. В одном варианте осуществления ID являются просто URI, и, по соглашению, эти URI являются URN.

7.1.1.2. Атрибуты.

Атрибуты являются типизированными значениями. Атрибуты могут быть именованными или безымянными. Имя именованного атрибута является простой строкой или URI. Значение атрибута является простым типом (строка, целое число или массив байтов) или составным типом (список или массив). Атрибуты типа 'список' содержат неупорядоченный список именованных атрибутов. Атрибуты типа 'массив' содержат упорядоченный массив безымянных атрибутов.

Поле 'attributes' объекта является (возможно, пустой) неупорядоченной коллекцией именованных атрибутов.

7.1.1.3. Расширения.

Расширения это элементы, которые можно добавлять к объектам для переноса необязательных или обязательных дополнительных данных. Расширения являются типизированными и также имеют уникальные ID. Расширения могут быть внутренними или внешними.

7.1.1.3.1. Внутренние расширения.

Внутренние расширения содержатся в объекте, которые они расширяют. Они имеют флаг 'critical', который указывает, необходима ли реализация, которая использует объект, знать конкретный тип данных расширения для расширения. В одном варианте осуществления, если реализация встречает объект с критическим расширением, имеющим тип данных, который она не понимает, она может отбросить весь объект.

В одном варианте осуществления, ID внутреннего расширения должен быть локально-уникальным: объект не может содержать два расширения с одним и тем же ID, но возможно, что два разных объекта содержат расширения с одинаковым ID.

Поле 'extensions' объекта является (возможно, пустой) неупорядоченной коллекцией внутренних расширений.

7.1.1.3.2. Внешние расширения.

Внешние расширения не содержатся в объекте, который они расширяют. Они появляются независимо от объекта и имеют поле 'subject', которое содержит ID объекта, который они расширяют. В одном варианте осуществления, ID внешнего расширения должно быть глобально-уникальным.

7.1.2. Content.

В одном варианте осуществления, объект «контент» является "внешним" объектом. Его формат и хранение не подчиняется механизму DRM, но подчиняется подсистеме управления контентом приложения хоста (например, контент может представлять собой видео-файл MP4, музыкальный трек MP3 и т.д.). В одном варианте осуществления, формат контента нуждается в обеспечении поддержки связывания ID с данными полезной нагрузки контента. Полезная нагрузка контента шифруется независимо от формата (обычно симметричным шифром, например AES).

7.1.3. ContentKey.

Объект ContentKey представляет уникальный ключ шифрования и связывает с ним ID. ID предназначен для того, чтоб объекты Protector и объекты Controller могли ссылаться на объекты ContentKey. Данные фактического ключа, инкапсулированные в объекте ContentKey, сами зашифрованы, в результате

чего их могут прочитать только получатели, авторизованные на дешифрование контента. Объект ContentKey указывает, какая криптосистема использовалась для шифрования данных ключа. Криптосистема, используемая для защиты данных ключа контента, называется системой распространения ключей. Можно использовать разные системы распространения ключей. Примером системы распространения ключей является вышеописанная система распространения ключей Scuba.

7.1.4. Protector.

Объект Protector содержит информацию, которая позволяет определять, какой ключ использовался для шифрования данных объектов Content. Он также содержит информацию, какой алгоритм шифрования использовался для шифрования этих данных. В одном варианте осуществления, объект Protector содержит один или несколько ID, которые являются ссылками на объекты Content, и в точности один ID, который является ссылкой на объект ContentKey, который представляет ключ, который использовался для шифрования данных. Если Protector указывает на более чем один объект Content, то все эти объекты Content представляют данные, которые были зашифрованы с использованием одного и того же алгоритма шифрования и одного и того же ключа. В одном варианте осуществления, пока используемая криптосистема позволяет безопасно использовать один и тот же ключ для разных элементов данных, не рекомендуется, чтобы объект Protector указывал на более чем один объект Content.

7.1.5. Control.

Объект управления содержит информацию, которая позволяет механизму DRM принимать решения относительно того, следует ли разрешить определенные действия над контентом, по запросу приложения хоста. В одном варианте осуществления, правила, которые регламентируют использование ключей контента, закодированы в объекте управления в виде байт-кода для выполнения виртуальной машиной. Объект управления также имеет уникальный ID, что позволяет объекту «контроллер» ссылаться на него. В одном варианте осуществления, объекты управления подписаны, благодаря чему механизм DRM может удостовериться, что байт-код объекта управления является действительным и доверенным, прежде чем использовать его для принятия решений. Действительность объекта управления также, в необязательном порядке, можно выводить путем проверки защищенного хэша, содержащегося в объекте «контроллер».

7.1.6. Controller.

Объект «контроллер» содержит информацию, которая позволяет механизму DRM определять, какой объект управления регламентирует использование одного или нескольких ключей, представленных объектами ContentKey. Объект «контроллер» содержит информацию, которая привязывает его к объектам ContentKey и объекту управления, на который он ссылается. В одном варианте осуществления, объекты «контроллер» подписываются (например, приложением упаковщика, который имеет сертификат, позволяющий ему подписывать объекты «контроллер»), что позволяет устанавливать действительность связи между ContentKey и объектом управления, который управляет им, а также действительность связи между ContentKey ID и данными фактического ключа. Подпись объекта «контроллер» может представлять собой подпись открытым ключом или подпись симметричным ключом или их комбинацию. Кроме того, когда дайджест объекта управления, на который ссылается объект «контроллер», включен в объект «контроллер», действительность объекта управления можно вывести без необходимости отдельно проверять подпись объекта управления.

7.1.6.1. Подпись симметричным ключом.

В одном варианте осуществления это предпочтительный тип подписи для объектов «контроллер», и она реализуется путем вычисления Кода аутентификации сообщения (MAC) объекта «контроллер», снабженного тем же ключом, который представлен соответствующим объектом ContentKey. В одном варианте осуществления канонический метод для этого MAC предусматривает использование HMAC с тем же алгоритмом хэширования, который был выбран в качестве алгоритма подписи PKI, используемого в той же конфигурации.

7.1.6.2. Подпись открытым ключом.

Этот тип подписи используется, когда нужно знать идентичность сущности, подписывающей объект «контроллер». Этот тип подписи реализуется посредством алгоритма подписи открытым ключом, подписывания секретным ключом принципала, который утверждает действительность этого объекта. В одном варианте осуществления при использовании этого типа подписи подпись симметричным ключом также будет присутствовать и подписывать как объект «контроллер», так и подпись открытым ключом, что позволяет гарантировать, что принципал, подписавший своим секретным ключом, также знает фактическое значение ключа контента, переносимого объектом ContentKey.

7.2. Объекты DRM идентичности и управления ключами.

Как описано выше объекты «узел» представляют сущности в профиле DRM, и никакой явной или неявной семантики не используется для определения того, что представляют объекты «узел». Данная конфигурация (профиль DRM) системы определяет, какие существуют типы принципалов, и какие роли и идентичности представляют разные объекты «узел». Эта семантическая информация обычно выражается с использованием атрибутов объекта «узел».

Объекты «связь» представляют соотношения между узлами. Объекты «связь» также, в необязатель-

ном порядке, могут содержать некоторые криптографические данные, что позволяет использовать связь для вывода ключа контента. Как и для узлов, в одном варианте осуществления, никакой явной или неявной семантики используется для определения, что означает соотношение связи means. В зависимости от того, что представляют начальный и конечный узлы связи в данном профиле DRM, смысл соотношения связи может выражать принадлежность, собственность, ассоциацию и/или многие другие типы соотношений. В типичном профиле DRM, некоторые объекты «узел» могут представлять пользователей, другие узлы могут представлять устройства, и прочие узлы могут представлять группы пользователей или авторизованные домены (AD). В таком контексте, связи между устройствами и пользователями могут представлять отношение собственности, и связи между пользователями и группами пользователей или доменами авторизации могут представлять отношения принадлежности. На фиг. 28B показана структура и взаимоотношение между узлами и связями в одном иллюстративном варианте осуществления.

7.2.1. Узел.

Объект «узел» представляет сущность в системе. Атрибуты объекта «узел» задают определенные аспекты того, что представляет объект «узел», например, роль или идентичность, представленную объектом «узел» в контексте профиля DRM. Объект «узел» также может иметь пару асимметричных ключей конфиденциальности, которая используется для нацеливания конфиденциальной информации на подсистемы, имеющие доступ к конфиденциальным частям объекта «узел» (обычно, сущность, представленную узлом, или некоторую сущность, которая отвечает за управление этим узлом). Конфиденциальная информация, нацеленная на узел, может быть зашифрована открытым ключом конфиденциальности этого узла. Объект «узел» также может иметь пару асимметричных ключей совместного пользования, и симметричным ключом совместного пользования могут совместно использовать объекты «связь», когда система использует Систему вывода ключа контента для распространения ключей контента, например, описанную здесь в другом месте. В предпочтительном варианте осуществления, только сущностям, подлежащим ссылке со стороны объектов «связь» или «управление», или сущностям, которым необходимо принимать криптографически нацеленную информацию, нужно иметь соответствующие объекты «узел».

7.2.2. Связь.

Объект «связь» - это подписанное утверждение, что существует ориентированное ребро в графе, вершинами которого являются объекты «узел». Для данного множества узлов и связей, мы говорим, что существует путь между узлом X и узлом Y, если существует ориентированный путь между вершиной узла X и вершиной узла Y в графе. Когда существует путь между узлом X и узлом Y, мы говорим, что узел Y доступен от узла X. Утверждения, представленные объектами «связь», используются для выражения, какие узлы доступны от других узлов. Объекты управления, которые управляют объектами «контент», могут проверить, прежде чем разрешить осуществление действия, что определенные узлы доступны от узла, связанного с сущностью, осуществляющей действие. Например, если узел D представляет устройство, которое хочет осуществить действие "воспроизведение" на объекте «контент», объект управления, который управляет объектом «контент», может проверить, доступен ли определенный узел U, представляющий определенного пользователя, от узла D. Чтобы определить, доступен ли узел U, механизм DRM может проверить, существует ли множество объектов «связь», которые могут установить путь между узлом D и узлом U.

В одном варианте осуществления, механизм DRM проверяет объекты «связь» прежде, чем использовать их для принятия решения о наличии путей в графе узлов. В зависимости от конкретных особенностей системы сертификатов (например, x509v3), используемой для подписывания объектов «связь», объектам «связь» можно назначать ограниченные сроки действия, отменять их и т.д. В одном варианте осуществления, механизм DRM непосредственно не имеет дела с политиками, определяющие, какие ключи могут подписывать объекты «связь», какие объекты «связь» можно создавать и срок действия объектов «связь». Вместо этого, эти политики используют информацию атрибутов узла. Для облегчения задачи применения определенных политик, в одном варианте осуществления предусмотрен способ расширения стандартных форматов сертификата с дополнительной проверкой ограничений. Эти расширения дают возможность выразить ограничения по действительности сертификатов для ключей, которые подписывают связи, благодаря чему ограничения, например, какие типы узлов соединяет связь, а также другие атрибуты, можно проверять прежде, чем признать связь действительной.

В одном варианте осуществления, объект «связь» может содержать объект управления, который будет использоваться для ограничения действительности связи. Кроме того, в одном варианте осуществления объект «связь» может содержать криптографические данные вывода ключа, которые снабжают пользователя ключами совместного пользования для распространения ключей. Эти криптографические данные содержат, помимо метаданных, личные и/или симметричные ключи совместного пользования начального узла, зашифрованные открытым ключом совместного пользования и/или симметричным ключом совместного пользования конечного узла.

7.3. Структуры данных.

В нижеследующих разделах описана более подробно иллюстративная модель объекта для объектов, рассмотренных выше, задающая поля, которые имеет объект каждого типа в одном иллюстративном варианте осуществления. Структуры данных описаны с использованием сравнительно простого синтаксиса

описания объекта. Каждый тип объекта задается классом, который может расширять родительский класс (т.е. отношение "это есть"). Описания классов приведены в терминах простых абстрактных типов "string" (строки символов), "int" (целочисленное значение), "byte" (8-битовое значение), и "boolean" (истина или ложь), но не определяют никаких конкретных правил кодирования ни для этих типов данных, ни для составных структур, содержащих эти типы. Способ кодирования или представления объектов может варьироваться в зависимости от реализации механизма. На практике, данный профиль использования механизма DRM может задавать, как представляются поля (например, с использованием схемы XML).

В одном иллюстративном варианте осуществления используются следующие обозначения:

class ClassName {
field1;
field2;
....
}
	Задает тип «класс». Тип «класс» это разнородный составной тип данных (также именуемый типом объекта). Этот составной тип состоит из одного или нескольких полей, каждое из которых имеет простой или составной тип. Каждое поле может быть отдельного типа.

type[]	Задает однородный составной тип данных (также именуемый типом списка или массива). Этот составной тип состоит из 0 или более элементов одного типа (0, когда список пуст).
String	Простой тип: представляет строку символов
Int	Простой тип: представляет целочисленное значение
Byte	Простой тип: представляет целочисленное значение от 0 до 255
Boolean	Простой тип: представляет логическое значение (истина или ложь)
class SubClass extends SuperClass {…}	Задает тип «класс», который расширяет другой тип «класс». Класс, который расширяет другой класс, содержит все поля расширяемого класса (именуемого суперклассом) помимо своих собственных полей
Abstract class {…}	Задает абстрактный тип «класс». Абстрактные типы «класс» это типы, которые можно расширять, но которые сами никогда не используются

{type field;}	Задает необязательное поле. Необязательное поле это поле, которое можно исключить из составного типа данных, который его содержит
(type field;)	Задает поле, которое будет пропущено при вычислении канонической последовательности байтов для заключения составного поля
class SubClass extends SuperClass(field=value) {…}	Задает подкласс типа «класс» и указывает, что для всех экземпляров этого подкласса, значение определенного поля суперкласса всегда равно фиксированному значению

7.3.1. Общие структуры.

В одном иллюстративном варианте осуществления используются следующие общие структуры:

```
abstract class Octobject {
   {string id;}
   Attribute[] attributes;
   InternalExtension[] extensions;
}

class Transform {
   string algorithm;
}

class Digest {
   Transform[] transforms;
   string algorithm;
   byte[] value;
}
class Reference {
   string id;
   {Digest digest;}
}
```

7.3.1.1. Атрибуты.

В одном варианте осуществления, существует четыре разновидности атрибутов: IntegerAttribute, StringAttribute, ByteArrayAttribute, и ListAttribute, каждый из которых имеет имя и тип.

```
abstract class Attribute {
   {string name;}
   string type;
}

class IntegerAttribute extends Attribute(type='int') {
   int value;
}

class StringAttribute extends Attribute(type='string') {
   string value;
}

class ByteArrayAttribute extends Attribute(type='bytes') {
   byte[] value;
}

Class ListAttribute extends Attribute(type='list') {
   Attribute[] attributes; // все должны быть именованными
}

Class ArrayAttribute extends Attribute(type='array') {
   Attribute[] attributes; // все должны быть безымянными
}
```

7.3.1.2. Расширения.

В рассматриваемом иллюстративном варианте осуществления, существует два типа расширений: внутренние расширения, которые переносятся внутри Octobject, и внешние расширения, которые переносятся вне Octobject.

```
abstract class ExtensionData {
string type;
}

abstract class Extension {
string id;
}

class ExternalExtension extends Extension {
string subject;
ExtensionData data;
}

class InternalExtension extends Extension {
boolean critical;
{Digest dataDigest;}
{ExtensionData data;}
}
```

В некоторых вариантах осуществления, важно иметь возможность проверять подпись объекта, даже если данная реализация не понимает конкретный тип ExtensionData. Таким образом, в одном варианте осуществления, добавляется уровень косвенности с полем dataDigest. Если спецификация ExtensionData требует, чтобы данные были частью подписи в контексте конкретного объекта, то поле dataDigest будет присутствовать. Реализация, которая понимает этот тип ExtensionData и потому способна вычислять его каноническое представление, может затем проверить дайджест. Если, в этом варианте осуществления, спецификация этого типа ExtensionData чтобы данные не были частью подписи, то поле dataDigest будет отсутствовать.

7.3.2. Объекты Node

```
class Node extends Octobject {

}
```

7.3.3. Объекты Link

```
class Link extends Octobject {
    string fromId;
    string toId;
    {Control control;}
}
```

7.3.4. Объекты Control

```
class Control extends Octobject {
    string protocol;
    string type;
    byte[] codeModule;
}
```

7.3.5. Объекты ContentKey

```
abstract class Key {
    string id;
    string usage;
    string format;
    byte[] data;
}

abstract class PairedKey extends Key {
    string pairId;
}


class ContentKey extends Octobject {
    Key secretKey;
}
```

В одном варианте осуществления, каждый ключ имеет уникальный id, формат, использование (которое может быть пустым), и данные. Поле 'usage', если оно не пусто, указывает, с какой целью можно использовать ключ. Для нормальных ключей контента, это поле пусто. Согласно вариантам осуществления, в которых используется схема распространения ключей, например, вышеописанная, это поле может указывать, является ли ключ ключом совместного пользования или ключом конфиденциальности. Поле 'format' указывает формат поля 'data' (например, 'RAW' для симметричных ключей или 'PKCS#8' для секретных ключей RSA и т.д.). Поле 'data' содержит данные фактического ключа, форматированные в соответствии с полем 'format'.

Для ключей, которые входят в состав пары ключей (например, ключей RSA), дополнительное поле 'pairId' дает уникальный идентификатор пары, что позволяет ссылаться на пару из других структур данных.

В одном варианте осуществления поле данных в объекте «ключ» является незашифрованным значением фактического ключа (т.е. это незашифрованное значение ключа, которое будет хэшировано), даже если фактическое представление объекта содержит зашифрованную копию ключа.

7.3.6. Объекты Controller.

```
class Controller extends Octobject {
    Reference controlRef;
    Reference[] contentKeyRefs;
}
```

8. Виртуальная машина.

Предпочтительные варианты осуществления описанного здесь механизма DRM используют виртуальную машину (иногда именуемую здесь "виртуальной машиной управления", "VM управления" или просто "VM") для выполнения программ управления, которые регламентируют доступ к контенту. Ниже описаны иллюстративные варианты осуществления такой виртуальной машины, но этот иллюстративный вариант осуществления допускает различные модификации и изменения конструкции. Описано также объединение иллюстративного варианта осуществления виртуальной машины (именуемой здесь виртуальной машиной "Plankton") с иллюстративным вариантом осуществления механизма DRM (именуемого здесь "Octopus"). Очевидно, однако, что варианты осуществления механизма управления цифровыми правами, архитектура и другие описанные здесь системы и способы можно использовать с любой подходящей виртуальной машиной или, в некоторых вариантах осуществления, вообще без виртуальной машины, и, таким образом, очевидно, что представленные ниже детали, касающиеся иллюстративных вариантов осуществления виртуальной машины, предназначены для иллюстрации, но не ограничения.

В предпочтительном варианте осуществления, VM управления это традиционная виртуальная машина, которую легко реализовать с использованием различных языков программирования с очень малым объемом кода. Она основана на простом наборе команд со стековой организацией, построенном исходя из минимализма, без чрезмерной концентрации на скорости выполнения или плотность кода. В случаях, когда требуется компактный код, можно использовать методы сжатия данных для сжатия байт-кода виртуальной машины.

В предпочтительных вариантах осуществления виртуальная машина управления призвана быть пригодной в качестве конечного продукта для языков программирования низкого или высокого уровня, и поддерживает ассемблер, C и FORTH. Кроме того, очевидно, что компиляторы для других языков, например, Java или специализированных языков, можно создавать относительно прямым путем для компиляции кода в формат (например, байт-код), используемый виртуальной машиной. В одном варианте осуществления виртуальная машина управления предназначена для размещения в среде хоста, а не для выполнения непосредственно на процессоре или в логических устройствах. В предпочтительных вариантах осуществления, естественной средой хоста для виртуальной машины является механизм DRM, хотя очевидно, что описанную здесь архитектуру виртуальной машины можно, альтернативно или дополнительно, использовать в других контекстах.

На фиг. 29 показана операционная среда иллюстративной реализации виртуальной машины управления 2902. Согласно фиг. 29 в одном варианте осуществления виртуальная машина 2902 выполняется в контексте своей среды хоста 2904, которая реализует некоторые функции, необходимые виртуальной машине для выполнения программ 2906. Обычно VM управления выполняется в механизме DRM 2908, который реализует ее среду хоста. Согласно фиг. 29, в предпочтительной базе данных, виртуальная машина 2902 и механизм DRM 2908 обращаются к защищенной базе данных 2910 для постоянного хранения информации состояния.

8.1. Архитектура.

8.1.1. Модель выполнения.

В предпочтительных вариантах осуществления, VM выполняет программы путем выполнения команд, записанных в виде байт-кода в кодовых модулях. Некоторые из этих команд могут вызывать функции, реализованные вне самой программы, осуществляя системные вызов. Системные вызовы можно реализовать на самой VM или делегировать среде хоста.

В одном варианте осуществления VM выполняет команды, записанные в кодовых модулях, в виде потока байт-кодов, загружаемых в память. VM поддерживает виртуальный регистр, именуемый Program Counter (PC) [счётчик команд], который увеличивается по мере выполнения команд. VM выполняет все команды по очереди, пока не встретит команду OP_STOP, команду OP_RET с пустым стеком вызова, или не возникнет исключение среды выполнения. Безусловные переходы указаны либо как относительный переход (указанный в виде байтового сдвига от текущего значения PC), либо как абсолютный адрес.

8.1.2. Модель памяти.

В одном варианте осуществления VM использует сравнительно простую модель памяти, в которой память делится на память для хранения данных и кодовую память. Например, память для хранения дан-

ных можно реализовать как единое, плоское, непрерывное пространство памяти, начинающееся по адресу 0, и можно реализовать как массив байтов, выделенный в динамической памяти приложения хоста или среды хоста. В одном варианте осуществления, попытки обратиться к памяти за пределами выделенного пространства будут приводить к исключению среды выполнения, которое приведет к завершению выполнения программы.

Память для хранения данных может распределяться между несколькими кодовыми модулями, одновременно загружаемыми виртуальной машиной. К данным в памяти для хранения данных можно обращаться посредством команд обращения к памяти, которые, в одном варианте осуществления, могут обеспечивать 32-разрядный или 8-разрядный доступ. 32-разрядные обращения к памяти осуществляются с использованием обратного порядка следования байтов. В предпочтительном варианте осуществления, не делается никаких предположений относительно выравнивания между памятью, которую видит виртуальная машина, памятью, управляемой хостом (т.е. виртуальной или физической памятью ЦП хоста).

В одном варианте осуществления, кодовая память представляет собой плоское, непрерывное пространство памяти, начинающееся по адресу 0, и можно реализовать как массив байтов, выделенный в динамической памяти приложения хоста или среды хоста.

VM может поддерживать загрузку более одного кодового модуля. Если VM загружает несколько кодовых модулей, в одном варианте осуществления все кодовые модули совместно используют одну и ту же память для хранения данных (хотя данные каждого модуля предпочтительно загружать по разным адресам), но каждый из них имеет свою собственную кодовую память, что не дает команде перехода в одном кодовом модуле вызвать переход к коду в другом кодовом модуле.

8.1.3. Стек данных.

В одном варианте осуществления, VM имеет так называемый стек данных, который представляет 32-битовые ячейки данных, хранящиеся в памяти для хранения данных. VM поддерживает виртуальный регистр, именуемый Stack Pointer [Указатель стека] (SP). После сброса, SP указывает на конец памяти для хранения данных, и стек растет вниз (при проталкивании данных в стек данных, регистр SP уменьшается). 32-битовые ячейки данных в стеке интерпретируются либо как 32-битовые адреса, либо как 32-битовые целые, в зависимости от команды, ссылающейся на данные стека. Адреса являются беззнаковыми целыми. В одном варианте осуществления, все остальные 32-битовые целочисленные значения в стеке данных интерпретируются как знаковые целые, если не указано обратное.

8.1.4. Стек вызова.

В одном варианте осуществления, VM управляет стеком вызова, который используется для вызова подпроцедур. В одном варианте осуществления, команды обращения к памяти не могут непосредственно считывать или записывать значения, протолкнутые в этот стек. Этот стек используется внутри VM при выполнении команд OP_JSR, OP_JSRR и OP_RET. Для данной реализации VM, размер этого стека адресов возврата может быть задан минимальным, в результате чего будет разрешено лишь определенное количество вложенных вызовов.

8.1.5. Псевдорегистры.

В одном варианте осуществления, VM резервирует небольшое пространство адресов в начале памяти для хранения данных для отображения псевдорегистров. В одном варианте осуществления, адреса этих псевдорегистров фиксированы. Например, можно задать следующие регистры:

Адрес	Размер	Имя	Описание
0	4	ID	32-битовый ID выполняющегося в данный момент сегмента кода. VM выбирает этот ID при загрузке модуля. VM изменяет этот регистр при переходе от сегмента кода одного модуля к сегменту кода другого модуля
4	4	DS	32-битовое значение, заданное равным абсолютному адресу данных, по которому загружен сегмент данных выполняющегося в данный

			момент модуля. Это значение определяется загрузчиком модулей VM
8	**4**	**CS**	32-битовое значение, заданное равным абсолютному адресу кода, по которому загружен сегмент кода выполняющегося в данный момент модуля. Это значение определяется загрузчиком модулей VM
12	**4**	**UM**	32-битовое значение, заданное равным абсолютному адресу данных первого байта после области пространства памяти для хранения данных, куда загружены сегменты данных кодовых модулей.

8.1.6. Карта памяти.

Ниже показана компоновка памяти для хранения данных и кодовой памяти в иллюстративном варианте осуществления.

Память для хранения данных

Диапазон адресов	**Описание**
0-15	Псевдорегистры
16-127	Зарезервирован для будущего использования VM/системой
128-255	Зарезервирован для использования приложением
256-DS-1	Не задан. VM может загружать сегменты данных кодовых модулей по любому адресу от 256 и выше. Если она выбирает адрес больше 256, использование пространства адресов от 256 до DS остается незаданным. Это означает, что реализация виртуальной машины может использовать его по своему усмотрению.
DS-UM-1	Образ сегментов данных одного или нескольких кодовых модулей, загруженных виртуальной машиной.
UM-Конец	Совместно используемое пространство адресов. Данные кодовых модулей и стек данных совместно используют это пространство. Стек данных располагается в конце этого пространства и растет вниз. Конец представляет последний адрес пространства памяти для хранения данных. Размер пространства памяти для хранения данных задается VM на основании требований к памяти, содержащихся в кодовом модуле, и требований реализации

Кодовая память.

Диапазон адресов	Описание
0-CS-1	Не задан. Виртуальная машина может загружать сегменты кода кодовых модулей по любому адресу от 0 и выше. Если она выбирает адрес больше 0, использование пространства адресов от 0 до CS остается незаданным. Это означает, что виртуальная машина может использовать его по своему усмотрению.
CS-CS+размер (сегмент кода)-1	Образ сегмента кода кодового модуля, загруженного виртуальной машиной

8.1.7. Выполнение процедур.

До выполнения кодовой процедуры, в одном варианте осуществления реализация виртуальной машины сбрасывает указатель стека данных, чтобы он указывал вершину инициализированного стека данных. Инициализированный стек данных содержит входные данные процедуры и доходит до конца памяти для хранения данных. Инициализированный стек данных можно использовать для передачи входных аргументов процедуре. В отсутствие инициализированного стека данных, указатель стека данных указывает на конец памяти для хранения данных. В одном варианте осуществления, начальный стек вызова либо пуст, либо содержит единый конечный адрес возврата, указывающий на команду OP_STOP, что принудительно прекращает выполнение процедуры по команде OP_STOP в случае, когда процедура заканчивается командой OP_RET.

Когда выполнение останавливается, либо вследствие выполнения конечной команды OP_RET с пустым стеком вызова, либо вследствие выполнения конечной команды OP_STOP, любые данные, оставшиеся в стеке данных, считаются выходом процедуры.

8.1.8. Исключения среды выполнения.

В одном варианте осуществления, любое из следующих условий считается исключением среды выполнения, которое приводит к немедленной остановке выполнения:

попытка обращения к памяти для хранения данных за пределами текущего пространства адресов памяти для хранения данных;

попытка задать PC равным или вызвать переход PC к адресу кода за пределами текущего пространства адресов кодовой памяти;

попытка выполнения незаданного байт-кода;

попытка выполнения команды OP_DIV с операндом «вершина стека», равным 0;

попытка выполнения команды OP_MOD с операндом «вершина стека», авным 0;

переполнение или опустошение стека вызова.

8.2. Набор команд.

В одном варианте осуществления, VM управления использует сравнительно простой набор команд. Хотя и ограниченный, этот набор команд достаточен для выражения программ любой сложности. Команды и их операнды представлены потоком байт-кодов. В одном варианте осуществления, набор команд работает со стеком, и, за исключением команды OP_PUSH, ни одна из команд не имеет прямых операндов. Операнды считываются из стека данных, и результаты проталкиваются в стек данных. В одном варианте осуществления, VM является 32-разрядной VM: все команды работают с 32-разрядными стековыми операндами, представляющими либо адреса памяти, либо знаковые целые. Знаковые целые представлены двоичным кодированием с дополнением двойки. Иллюстративный вариант осуществления набора команд для использования с VM управления показан в нижеследующей таблице. В таблице, стековые операнды для команд с двумя операндами представлены как "A, B", где операнд на вершине стека указан последним (т.е. "B"). Если не указано обратное, термин "проталкивание", используемый в нижеследующем описании одного иллюстративного варианта осуществления, означает продвижение 32-битового значения к вершине стека данных.

Код операции	Имя	Байт-код	Операнды	Описание
OP_NOP	Нет операций	0		Ничего не делает
OP_PUSH	Проталкивание константы	1	N (прямой)	Проталкивает 32-битовую константу
OP_DROP	Сброс	2		Удаляет верхнюю ячейку стека данных
OP_DUP	Дублирование	3		Дублирует верхнюю ячейку стека данных
OP_SWAP	Обмен	4		Меняет местами две ячейки стека данных
OP_ADD	Сложение	5	A, B	Проталкивает сумму A и B (A+B)
OP_MUL	Умножение	6	A, B	Проталкивает произведение A и B (A*B)
OP_SUB	Вычитание	7	A, B	Проталкивает разность между A и B (A-B)
OP_DIV	Деление	8	A, B	Проталкивает частное от деления A на B (A/B)
OP_MOD	Модуль	9	A, B	Проталкивает A по модулю B (A%B)
OP_NEG	Отрицание	10	A	Проталкивает отрицание с дополнением 2 для A (-A)
OP_CMP	Сравнение	11	A, B	Проталкивает -1, если A меньше B, 0, если A равно B, и 1, если A больше B
OP_AND	И	12	A, B	Проталкивает побитовое И для A и B (A & B)
OP_OR	Или	13	A, B	Проталкивает побитовое ИЛИ для A и B (A | B)
OP_XOR	Исключающее ИЛИ	14	A, B	Проталкивает побитовое исключающее ИЛИ для A и B (A ^ B)
OP_NOT	Логическое отрицание	15	A	Проталкивает логическое отрицание A (1, если A равно 0, и 0, если A не равно 0)
OP_SHL	Сдвиг влево	16	A, B	Проталкивает A, логически сдвинутое влево на B битов (A << B)

OP_SHR	Сдвиг вправо	17	A, B	Проталкивает A, логически сдвинутое вправо на B битов (A >> B)
OP_JMP	Переход	18	A	Переход к A
OP_JSR	Переход к подпроцедуре	19	A	Переход к подпроцедуре по абсолютном адресу A. Текущее значение PC проталкивается в стек вызова.
OP_JSRR	Переход к подпроцедуре (относительный)	20	A	Переход к подпроцедуре по адресу PC+A. Текущее значение PC проталкивается в стек вызова.
OP_RET	Возврат из подпроцедуры	21		Возврат из подпроцедуры по адресу возврата, вытолкнутому из стека вызова.
OP_BRA	Переход всегда	22	A	Переход к PC + A
OP_BRP	Переход при положительном значении	23	A, B	Переход к PC+B, если A > 0
OP_BRN	Переход при отрицательном значении	24	A, B	Переход к PC+B, если A < 0
OP_BRZ	Переход при нулевом значении	25	A, B	Переход к PC+B, если A равно 0
OP_PEEK	Считывание	26	A	Проталкивает 32-битовое значение по адресу A
OP_POKE	Запись	27	A, B	Сохраняет 32-битовое значение A по адресу B
OP_PEEKB	Считывание байта	28	A	Считывает 8-битовое значение по адресу A, дополняет его нулями до 32 битов и проталкивает его в стек данных
OP_POKEB	Запись байта	29	A, B	Сохраняет 8 младших битов значения A по адресу B
OP_PUSHSP	Проталкивание указателя стека	30		Проталкивает значение SP
OP_POPSP	Выталкивание указателя стека	31	A	Задает значение SP равным A
OP_CALL	Системный вызов	32	A	Осуществляет системный вызов с индексом A
OP_STOP	Останов	255		Прекращает выполнение

8.3. Кодовые модули.

В предпочтительном варианте осуществления кодовые модули хранятся в атомарном формате, аналогичном или идентичном тому, который используется для файлов MPEG-4, и атомы содержат 32-битовый размер (например, представленный 4 байтами в обратном порядке следования байтов), после которого следует 4-байтовый тип (например, байты, соответствующие значениям ASCII для букв алфавита), после которого следует полезная нагрузка (например, 8 байтов).

На фиг. 30 показан формат иллюстративного кодового модуля 3000. Согласно фиг. 30, атом pkCM 3002 является атомом верхнего уровня кодового модуля. Он содержит последовательность податомов. В одном варианте осуществления, атом pkCM 3002 содержит один атом pkDS 3004, один атом pkCS 3006, один атом pkEX 3008, и, возможно, один атом pkRQ 3010. Фтом pkCM 3002 также может содержать любое количество других атомов, которые, в одном варианте осуществления, игнорируются, если присутствуют. В одном варианте осуществления, порядок податомов не указан, поэтому реализации не обязаны предусматривать конкретный порядок.

8.3.1. Атом pkDS.

Согласно фиг. 30, атом pkDS 3004 содержит образ памяти 3005 сегмента данных, который может загружаться в память для хранения данных. Согласно фиг. 31A, в одном варианте осуществления образ памяти 3005 представлен последовательностью байтов 3112, состоящей из одного байта заголовка 3114, после которого следует ноль или более байтов данных 3116. Байт заголовка 3114 кодирует номер версии, который идентифицирует формат следующих байтов 3116.

В одном варианте осуществления, задается только один номер версии (т.е. DataSegmentFormatVersion=0), и в этом формате байты данных образа памяти представляют необработанный образ, подлежащий загрузке в память. Загрузчик виртуальной машины загружает только байты данных 3116 образа памяти 3105, не включающие в себя байт заголовка 3114. В одном варианте осуществления, загрузчик виртуальной машины способен отказываться загружать образ в любом другом формате.

8.3.2. Атом pkCS.

Согласно фиг. 30, атом pkCS 3006 содержит образ памяти 3007 сегмента кода, который может загружаться в кодовую память. Согласно фиг. 31B в одном варианте осуществления образ памяти 3007 представлен последовательностью байтов 3120, состоящей из одного байта заголовка 3122, после которого следует ноль или более байтов данных 3124. Байт заголовка 3122 кодирует номер версии, который идентифицирует формат следующих байтов 3124.

В одном варианте осуществления задается только один номер версии (т.е. CodeSegmentFormatVersion=0), и, согласно фиг. 31C, в этой версии байт, следующий за байтом заголовка 3122, содержит еще один байт заголовка 3130, содержащий номер версии, который идентифицирует кодирование в виде байт-кода следующих байтов 3132. В примере, показанном на фиг. 31C, байт заголовка 3130 идентифицирует ByteCodeVersion=0, который указывает, что байты данных 3132 содержат необработанную последовательность байтов со значениями байт-код, например, заданными в вышеописанном иллюстративном наборе команд. В предпочтительном варианте осуществления, загрузчик виртуальной машины загружает только часть 3132 байт-кода, состоящую из байтов данных, но не два байта заголовка 3122, 3130.

8.3.3. Атом pkEX.

Согласно фиг. 30, атом pkEX 3008 содержит список элементов экспорта. В примере, показанном на фиг. 30, первые четыре байта 3009 атома pkEX 3008 кодируют 32-битовое беззнаковое целое в обратном порядке следования байтов, равное количеству последующих элементов. Согласно фиг. 31D, каждый следующий элемент экспорта 3160 состоит из имени, кодированного одним байтом 3162, содержащим размер имени, S, после которого следует S байтов 3164, содержащих ASCII-символы имени, включая оконечный нуль 3166, после которых следует 32-битовое беззнаковое целое 3168 в обратном порядке следования байтов, представляющее байтовый сдвиг именованной точки входа, измеряемый от начала данных байт-кода, хранящихся в атоме pkCS. На фиг. 31E показан пример элемент 3170 таблицы экспорта для точки входа MAIN со сдвигом 64, в котором первый байт 3172 указывает, что размер имени (т.е. "MAIN"), плюс оконечный нуль, равен пяти байтам, и в котором последние четыре байта 3174 указывают, что байтовый сдвиг равен 64.

8.3.4. Атом pkRQ.

Согласно фиг. 30, атом pkRQ 3010 содержит требования, которым должна удовлетворять реализация виртуальной машины для выполнения кода в кодовом модуле. В одном варианте осуществления, этот атом является необязательным, и, если он отсутствует, виртуальная машина использует настройки реализации, принятые по умолчанию, которые, например, могут быть заданы профилем реализации.

В одном варианте осуществления, атом pkRQ состоит из массива 32-битовых беззнаковых целочисленных значений, по одному на каждое поле:

Имя поля	Описание
vmVersion	ID версии спецификации VM
minDataMemorySize	Минимальный размер в байтах памяти для хранения данных, доступной коду. Она включает в себя память для хранения данных, используемую для загрузки образа сегмента данных, а также память для хранения данных, используемую стеком данных. В одном варианте осуществления, VM должна отказываться загружать модуль, если он не может удовлетворить этому требованию
minCallStackDepth	Минимальное количество вложенных вызовов подпроцедуры (OP_JSR и OP_JSRR), которое должна поддерживать VM. В одном варианте осуществления, VM должна отказываться загружать модуль, если он не может удовлетворить этому требованию.
Flags	Набор битовых флагов, представляющих необходимые особенности VM. В одном варианте осуществления, реализация VM должна отказываться загружать кодовый модуль, в котором задан какой-либо неизвестный флаг. Например, если не задано ни одного флага, в одном варианте осуществления, реализация VM должна проверить, что это поле задано равным 0.

8.3.5. Загрузчик модулей.

Виртуальная машина отвечает за загрузку кодовых модулей. При загрузке кодового модуля, образ памяти сегмента данных, закодированный в атоме pkDS, загружается по адресу памяти в память для хранения данных. Этот адрес выбирает загрузчик VM, и он хранится в псевдорегистре DS при выполнении кода.

При загрузке кодового модуля выполняется особая процедура под названием "Global.OnLoad", если эта процедура найдена среди элементов таблицы экспорта. Эта процедура не берет никаких аргументов в стеке и возвращает целочисленный статус по возвращении, причем 0 обозначает успех, и отрицательный код ошибки обозначает условие ошибки.

При выгрузке кодового модуля (или когда виртуальная машина, загрузившая модуль, избавляется от него), выполняется особая процедура под названием «Global.OnUnload", если эта процедура найдена

среди элементов таблицы экспорта. Эта процедура не берет никаких аргументов в стеке, и возвращает целочисленный статус по возвращении, причем 0 обозначает успех, и отрицательный код ошибки обозначает условие ошибки.

8.4. Системные вызовы.

Программы виртуальной машины могут вызывать функции, реализованный вне сегмента кода ее кодовых модулей. Это делается с использованием команды OP_CALL, которая берет целочисленный стековый операнд, указывающий номер системного вызова, который должен быть сделан. В зависимости от системного вызова, реализация может представлять собой процедуру байт-кода в другом кодовом модуле (например, библиотеке функций утилит), исполняемую непосредственно на VM в родном формате реализации VM, или делегируемую внешнему программному модулю, например, среде хоста VM.

В одном варианте осуществления, если команда OP_CALL выполняется с операндом, который содержит число, которое не соответствует никакому системному вызову, VM ведет себя так, как если бы был сделан системный вызов SYS_NOP.

8.4.1. Выделение номеров системного вызова.

В рассматриваемом иллюстративном варианте осуществления, номера системного вызова от 0 по 1023 зарезервированы для фиксированных системных вызовов (эти системные вызовы будут иметь один и тот же номер во всех реализациях VM). Номера системного вызова с 1024 по 16383 доступны VM для динамического присвоения (например, номера системного вызова, возвращаемые System.FindSystem CallByName, могут динамически выделяться VM и не обязаны быть одинаковыми номерами во всех реализациях VM).

В одном иллюстративном варианте осуществления указаны следующие фиксированные номера системного вызова:

Мнемокод	Номер	Системный вызов
SYS_NOP	0	System.NoOperation
SYS_DEBUG_PRINT	1	System.DebugPrint
SYS_FIND_SYSTEM_CALL_BY_NAME	2	System.FindSystemCallB yName
SYS_SYSTEM_HOST_GET_OBJECT	3	System.Host.GetObject
SYS_SYSTEM_HOST_SET_OBJECT	4	System.Host.SetObject

8.4.2. Стандартные системные вызовы.

В одном варианте осуществления поддерживается несколько стандартных системных вызовов, которые полезны для записи программ управления. Эти вызовы включают в себя системные вызовы с фиксированными номерами, перечисленные в вышеприведенной таблице, а также системные вызовы, имеющие динамически определяемые номера (т.е. их номера системного вызова извлекаются путем осуществления системного вызова System.FindSystemCallByName, которому в качестве аргумента передаются их имена).

В одном варианте осуществления системные вызовы, указанные в этом разделе, которые могут возвращать отрицательный код ошибки, могут возвращать коды ошибки с любым отрицательным значением. В разделе 8.4.4 указаны конкретные иллюстративные значения. В одном варианте осуществления, если возвращаются отрицательные значения кода ошибки, которые заранее не заданы, они интерпретируются как общее значение кода ошибки FAILURE.

System.NoOperation. Этот вызов не берет никаких аргументов и ничего не возвращает, и просто завершается, ничего не сделав. Он используется, в основном, для тестирования VM.

System.DebugPrint. Этот вызов берет в качестве аргумента из вершины стека адрес ячейки памяти, содержащей строку, оканчивающуюся символом конца строки, и ничего не возвращает. Вызов этой функции приводит к печати строки текста в выходе отладки, что может быть полезно при отладке. Если реализация VM не предусматривает возможности вывода текста отладки (что, например, может иметь место в среде без разработки), VM может игнорировать вызов и интерпретировать его как System.NoOperation.

System. FindSystemCallByName. Этот вызов находит номер системного вызова по его имени. Вызов берет в качестве аргумента (из вершины стека) адрес строки ASCII, оканчивающейся символом конца строки, содержащей имя искомого системного вызова, и возвращает (в вершину стека) номер системного вызова, если системный вызов с указанным именем реализован, ERROR_NO_SUCH_ITEM, если системный вызов не реализован, и отрицательный код ошибки в случае возникновения ошибки.

System.Host.GetLocalTime. Этот вызов не берет никаких аргументов и возвращает, в вершину стека, текущее значение местного времени хоста, которое, в одном варианте осуществления, выражается 32-битовым знаковым целым, равным числу минут, прошедших с 00:00:00 1 января 1970 г., или отрицательным кодом ошибки.

System.Host.GetLocalTimeOffset. Этот вызов не берет никаких аргументов и возвращает, в вершину

стека, текущее смещение по времени (относительно времени UTC) хоста, которое, в одном варианте осуществления, выражается 32-битовым знаковым целым, равным числу минут, составляющему разницу между местным временем и временем UTC (т.е. LocalTime - UTC).

System.Host.GetTrustedTime. Этот вызов не берет никаких аргументов и возвращает, в вершину стека, доверенное время и значение одного или нескольких флагов. В одном варианте осуществления, доверенное время это текущее значение доверенных часов (если система включает в себя такие доверенные часы), или отрицательный код ошибки, если доверенное время недоступно. В одном варианте осуществления, значение доверенного времени выражается 32-битовым знаковым целым, равным числу минут, прошедших с 00:00:00 1 января 1970 г. согласно UTC, или отрицательным кодом ошибки. В одном варианте осуществления, представляют собой набор битовых флагов, которые дополнительно задают текущее состояние доверенных часов. В одном варианте осуществления, в случае возникновения ошибки (например, значение TrustedTime равно отрицательному коду ошибки) возвращается значение флагов, равное 0.

В одном варианте осуществления, задается следующий флаг:

Битовый индекс (0 это LSB)	Имя	Описание
0	TIME_IS_ESTIMATE	Известно, что значение TrustedTime не является самым точным его значением, и поэтому должно рассматриваться как оценка.

Этот системный вызов имеет смысл в системах, где реализованы доверенные часы, которые можно синхронизировать с доверенным источником времени, и поддерживается монотонный счетчик времени. Не гарантируется, что значение доверенного времени всегда будет точным, но в одном варианте осуществления требуется наличие следующих свойств:

значение доверенного времени выражается как значение времени UTC (доверенное время не является местным временем, поскольку текущее местоположение обычно нельзя точно определить);

доверенное время никогда не идет назад;

доверенные часы идут не быстрее реального времени.

Поэтому, в этом иллюстративном варианте осуществления, значение TrustedTime находится между значением последнего синхронизированного времени (синхронизированного с доверенным источником времени) и текущим реальным временем. Если система способна определить, что ее доверенные часы действовали и обновлялись постоянно и нормально без перерыва с момента последней синхронизации с доверенным источником времени, она может определить, что значение TrustedTime является не оценочным, а точным значением, и задать флаг TIME_IS_ESTIMATE равным 0.

В одном варианте осуществления, если доверенные часы обнаруживают, что наступило условие аппаратного или программного сбоя, и они не могут возвратить даже оценку доверенного времени, возвращается код ошибки, и значение возвращаемых флагов задается равным 0.

System.Host.GetObject: Этот системный вызов является общим интерфейсом, который позволяет программе обращаться к объектам, обеспеченным хостом виртуальной машины. Вызов System.Host.GetObject берет следующие аргументы (перечисленные от вершины стека вниз): Parent, Name, ReturnBuffer и ReturnBufferSize. Где "Parent" это 32-битовый описатель родительского контейнера; "Name" это адрес строки, оканчивающейся символом конца строки, содержащей путь к запрашиваемому объекту относительно родительского контейнера; "ReturnBuffer" это адрес буфера памяти, где должно храниться значение объекта; и "ReturnBufferSize" это 32-битовое целое, указывающее размер в байтах буфера памяти, где должно храниться значение объекта.

Вызов System.Host.GetObject создает следующие выходы (перечисленные от вершины стека вниз): TypeID, Size. "TypeId" это id типа объекта или отрицательный код ошибки, если вызов не удается. Если запрашиваемый объект не существует, возвращается ошибка в виде ERROR_NO_SUCH_ITEM. Если буфер, предоставленный для возвращаемого значения, слишком мал, возвращается ошибка в виде ERROR_INSUFFICIENT_SPACE. Если часть дерева объектов, к которой происходит обращение, имеет управление доступом, и вызывающая программа не имеет разрешения на доступ к объекту, возвращается ERROR_PERMISSION_DENIED. Также могут возвращаться и другие коды ошибки. "Size" это 32-битовое целое, указывающее размер в байтах данных, возвращаемых в буфер, предоставленный вызывающей сущностью, или необходимый размер, если вызывающая сущность обеспечила слишком малый буфер.

В одном варианте осуществления существует четыре типа объектов хоста: строки, целые числа, массивы байтов и контейнеры.

Тип объекта	Имя Id типа	Значение Id типа
Контейнер	OBJECT_TYPE_CONTAINER	0
Целое	OBJECT_TYPE_INTEGER	1
Строка	OBJECT_TYPE_STRING	2
Массив байтов	OBJECT_TYPE_BYTE_ARRAY	3

В одном варианте осуществления значение объекта «массив байтов» представляет собой массив 8-битовых байтов, значение объекта «строка» представляет собой строку символов, оканчивающуюся символом конца строки, закодированную в UTF-8, и значение объекта «целое число» представляет собой 32-битовое знаковое целочисленное значение. Контейнеры представляют собой контейнеры общего вида, которые содержат последовательность из любого количества объектов любой комбинации типов. Объекты, содержащиеся в контейнере, называются детьми этого контейнера. Значение контейнера представляет собой 32-бтовый описатель контейнера, уникальный в данном экземпляре VM. В одном варианте осуществления, корневой контейнер '/' имеет фиксированное значение описателя 0.

В одном варианте осуществления пространство имен для объектов хоста имеет иерархическую структуру, где имя дочернего объекта контейнера строится путем присоединения имени дочернего объекта к имени родительского контейнера, каковые имена разделены символом '/'. Объекты «строка» и «целое число» не имеют детей. Например, если контейнер назван '/Node/Attributes' и имеет дочернюю строку под именем 'Type', то '/Node/Attributes/Type' относится к дочерней строке.

Корнем пространства имен является '/'. Все абсолютные имена начинаются с '/'. Имена, которые не начинаются с '/', являются относительными именами. Относительные имена относятся к родительскому контейнеру. Например, имя 'Attributes/Type', относительно родителя '/Node', это объект с абсолютным именем '/Node/Attributes/Type'.

В одном варианте осуществления, объекты «контейнер» также могут иметь реальные и виртуальные дочерние объекты, к которым можно обращаться с использованием виртуальных имен. Виртуальные имена это имена, которые не присоединены объектам хоста, но, по соглашению, идентифицируют либо безымянные дочерние объекты, дочерние объекты с другим именем, либо виртуальные дочерние объекты (дочерние объекты, которые не являются реальными детьми контейнера, но динамически создаются по запросу).

В одном варианте осуществления, для объектов, следующие виртуальные имена заданы как имена виртуальных дочерних объектов:

Виртуальное имя	Описание
@Name	Виртуальный объект «строка»: имя объекта. Если объект не имеет имени, значение является пустой строкой. Заметим, что безымянные объекты доступны только через @<n> виртуальное имя объекта «контейнер» (см. ниже)
@Size	Виртуальный объект «целое число». Целочисленное значение равно размеру в байтах, необходимому для сохранения этого объекта. Для целых чисел, это значение равно 4; для строк, это количество байтов, необходимое для сохранения строки в кодировке UTF-8 плюс пустой байт-знак конца строки. Для массивов байтов, это количество байтов в массиве
@Type	Виртуальный объект «целое число». Целочисленное значение равно Type Id объекта

Для контейнеров, следующие виртуальные имена заданы как имена виртуальных дочерних объектов в одном варианте осуществления:

	Виртуальное имя	Описание
Виртуальный индекс	@<n>	Виртуальный объект: <n>-й объект в контейнере. Первый объект в контейнере имеет индекс 0. <n> выражается десятеричным числом. Пример: если 'Attributes' это контейнер, который содержит 5 дочерних объектов, 'Attributes/@4' это 5-й дочерний объект контейнера.
Виртуальный размер	@Size	Виртуальный объект «целое число». Целочисленное значение равно количеству объектов в контейнере.

Примеры. В нижеследующей таблице приведен пример иерархии объектов Host:

Имя	**Значение**	**Дети**			
Node	1	**Имя**	**Значение**	**Дети**	
		Type	"Device"		
		Имя	**Значение**	**Дети**	
		Attributes	2	**Имя**	**Значение**	**Дети**
				Color	"Red"	
				Имя	**Значение**	**Дети**
				Size	78	
				Имя	**Значение**	**Дети**
				Domain	"TopLevel"	

В этом примере, вызов System.Host.GetObject (parent=0, name="Node") возвращает ID типа 0 (т.е. контейнер) и приводит к записи значения описателя 1 в буфер, предоставленный вызывающей сущностью. Размер значения равен 4 байтам.

Вызов System.Host.GetObject (parent=0, name="Node/Attributes/Domain") возвращает ID типа 2 (т.е. строку) и приводит к записи строки "TopLevel" в буфер, предоставленный вызывающей сущностью. Размер значения равен 9 байтам.

Вызов System.Host.GetObject (parent=1, name="Attributes/@1") возвращает ID типа 1 (т.е. целое) и приводит к записи целого числа 78 в буфер, предоставленный вызывающей сущностью. Размер значения равен 4 байтам.

Вызов System. Host.GetObject (parent=0, name="DoesNotExist") возвращает код ошибки ERROR_NO_SUCH_ITEM.

System.Host.SetObject. Этот системный вызов является общим интерфейсом, который позволяет программе создавать, записывать и уничтожать объекты, обеспеченные хостом виртуальной машины. Описание имен и типов объектов такое же, как для вышеописанного вызова System.Host.GetObject. Заметим, что объекты хоста поддерживают возможность своей записи или уничтожения, и не все контейнеры поддерживают создание дочерних объектов. Когда производится вызов SetObject объекта, который не поддерживает операцию, возвращается ERROR_PERMISSION_DENIED.

Системный вызов System.Host.SetObject берет в качестве аргументов следующие параметры, перечисленные от вершины стека вниз:

Вершина стека

Parent
Name
ObjectAddress
ObjectType
ObjectSize
…

Parent: 32-битовый описатель родительского контейнера.

Name: адрес строки, оканчивающейся символом конца строки, содержащей путь к объекту относительно родительского контейнера.

ObjectAddress: адрес буфера памяти, где хранится значение объекта. Если адрес равен 0, вызов интерпретируется как запрос на уничтожение объекта. Данные по адресу зависят от типа объекта.

Object Type: ID типа объекта.

ObjectSize: 32-битовое целое, указывающее размер в байтах буфера памяти, где хранится значение объекта. В рассматриваемом иллюстративном варианте осуществления, размер задан равным 4 для объектов «целое число» и размеру буфера памяти, с учетом пустого символа конца строки, для объектов «строка». Для объектов «массив байтов», размер равен количеству байтов в массиве.

Системный вызов System.Host.SetObject возвращает ResultCode в вершину стека в качестве выхода. ResultCode равен 0, если вызов успешен, и равен отрицательному коду ошибки, если вызов неудачен. Если вызов является запросом на уничтожение объекта, и запрашиваемый объект не существует, или вызов является запросом на создание или запись объекта, и родитель объекта не существует, возвращается код ошибки в виде ERROR_NO_SUCH_ITEM. Если часть дерева объектов, к которой происходит обращение, имеет управление доступом, и вызывающая программа не имеет разрешения на доступ к объекту, возвращается ERROR_PERMISSION_DENIED. Также могут возвращаться и другие коды ошибки.

Существует особый случай, когда объект ссылается на контейнер, и ObjectAddress не равен 0. В этом случае параметр ObjectSize задается равным 0, и значение ObjectAddress игнорируется. Если контейнер уже существует, ничего не происходит, и возвращается ResultCode равный SUCCESS. Если контейнер не существует, и родитель контейнера является записываемым, создается пустой контейнер.

Octopus.Links.IsNodeReachable. Этот системный вызов используется программами управления для проверки, доступен ли данный узел от узла, связанного с сущностью, вмещающей этот экземпляр виртуальной машины. Вызов берет в качестве аргумента NodeId из вершины стека, где NodeId это строка, оканчивающаяся символом конца строки, содержащая ID конечного узла, подлежащего тестированию на доступность. В качестве выхода, вызов возвращает ResultCode и StatusBlockPointer в вершину стека. ResultCode это целочисленное значение, равное 0, если узел доступен, или отрицательный код ошибки, если нет. StatusBlockPointer это адрес стандартного ExtendedStatusBlock, или 0, если не возвращается ни одного блока состояний.

System.Host.SpawnVm. Этот системный вызов используется программами управления для запроса на создание нового экземпляра виртуальной машины и на загрузку нового кодового модуля. В одном варианте осуществления, хост вновь созданной виртуальной машины предъявляет те же объекты хоста, которые предъявлялись вызывающей сущности, за исключением того, что объект хоста "/Octopus/Runtime/Parent/Id" задан равным идентификатору вызывающей сущности. В одном варианте осуществления, этот объект хоста является контейнером. Детьми этого контейнера являются объекты типа строка, каждый из которых имеет значение, представляющее имя. В одном варианте осуществления, семантика и конкретные детали этих имен указаны в спецификации хоста виртуальной машины.

В одном варианте осуществления, когда виртуальная машина, которая выполняет код для вызывающей сущности прекращает существование, любая порожденная виртуальная машина, которая не была в явном виде освобождена путем вызова System.Host.ReleaseVm, автоматически освобождается системой, как при вызове System.Host.ReleaseVm.

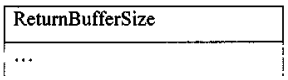
Вызов System.Host.SpawnVm берет в качестве аргумента ModuleId из вершины стека. ModuleId идентифицирует кодовый модуль, подлежащий загрузке в новый экземпляр виртуальной машины. В одном варианте осуществления, спецификация хоста виртуальной машины описывает механизм, позволяющий находить фактический кодовый модуль, соответствующий этому ID модуля.

Вызов System.Host.SpawnVm возвращает ResultCode и VmHandle в вершину стека. ResultCode это целочисленное значение, равное 0, если вызов успешен, и отрицательному коду ошибки, если он неудачен. VmHandle это целочисленное значение, идентифицирующее экземпляр виртуальной машины, который был создан. В случае неудачного вызова, этот описатель задается равным 0. В одном варианте осуществления, гарантируется лишь, что этот описатель будет уникальным в виртуальной машине, в которой делается этот вызов.

System.Host.CallVm. Этот системный вызов используется программами управления для вызова процедур, реализованных в кодовых модулях, загруженных в экземпляры виртуальной машины, созданные с использованием системного вызова System.Host.SpawnVm. Этот системный вызов берет следующие аргументы, начиная с вершины стека.

Вершина стека:

VmHandle
EntryPoint
ParameterBlockAddress
ParameterBlockSize
ReturnBufferAddress



VmHandle: целочисленное значение, представляющее описатель виртуальной машины, созданной путем вызова System.Host.SpawnVm.

EntryPoint: адрес строки, оканчивающейся символом конца строки, которая задает имя точки входа в вызов. Это имя должно совпадать с именем одной из точек входа в таблице экспорта кодового модуля, загруженного в экземпляр виртуальной машины, который соответствует параметру VmHandle.

ParameterBlockAddress: адрес блока памяти, который содержит данные, подлежащие передаче вызываемой сущности. Если вызываемой сущности не передаются никакие параметры, этот адрес задается равным 0.

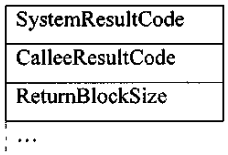
ParameterBlockSize: размер в байтах блока памяти по адресу ParameterBlockAddress или 0, если ParameterBlockAddress равен 0.

ReturnBufferAddress: адрес буфера памяти, где вызывающая сущность может принимать данные от вызываемой сущности. Если вызывающая сущность не ждет никаких данных от вызываемой сущности, этот адрес задается равным 0.

ReturnBufferSize: размер в байтах буфера памяти по адресу ReturnBufferAddress или 0, если ReturnBufferAddress равен 0.

Вызов System.Host.CallVm возвращает следующий выход в вершину стека:

Вершина стека:



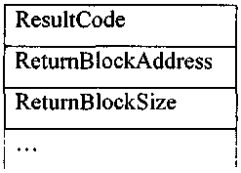
SystemResultCode: целочисленное значение, равное 0, если вызов увенчался успехом, или отрицательному коду ошибки в случае неудачи. Это значение определяется системой, а не вызываемой сущностью. Успех указывает лишь, что система смогла успешно найти процедуру для вызова, выполнить процедуру и получить возвращаемое значение от процедуры. Возвращаемое значение от самой роцедуры возвращается в виде значения CalleeResultCode.

CalleeResultCode: целочисленное значение, возвращаемое вызываемой сущностью.

ReturnBlockSize: размер в байтах данных, возвращаемых в буфер, предоставленный вызывающей сущностью, или необходимый размер, если вызывающая сущность обеспечила слишком малый буфер. Если вызываемая сущность не возвращает никаких данных, значени равно 0.

В рассматриваемом иллюстративном варианте осуществления, вызываемая процедура отвечает следующим соглашениям по интерфейсу: при вызове процедуры, вершина стека содержит значение ParameterBlockSize, обеспеченное вызывающей сущностью, указывающее размер блока параметра, следующего после байтов данных ParameterBlockSize. Если размер не кратен 4, данные в стеке дополняются нулями, чтобы указатель стека оставался кратным 4. По возвращении, вызываемая процедура помещает в стек следующие возвращаемые значения:

Вершина стека:



ReturnBlockAddress: адрес блока памяти, который содержит данные, возвращаемые вызывающей сущности. Если никаких данных не возвращается, этот адрес задается равным 0.

ReturnBlockSize: размер в байтах блока памяти по адресу ReturnBlockAddress или 0, если ReturnBlockAddress равен 0.

System.Host.ReleaseVm. Этот системный вызов используется программами управления для освобождения виртуальной машины, порожденной предыдущим вызовом System.Host.SpawnVm. Любые виртуальные машины, порожденные отпущенной виртуальной машиной, освобождаются, и т.д., рекурсивно. Вызов System.Host.ReleaseVm берет в качестве аргумента VmHandle из вершины стека, причем VmHandle представляет описатель виртуальной машины, созданной путем вызова System.Host.SpawnVm. Вызов System.Host.ReleaseVm возвращает в вершину стека ResultCode в качестве выхода. ResultCode это целочисленное значение, равное 0, если вызов увенчался успехом, или отрицательному коду ошибки в случае неудачи.

8.4.3. Стандартные структуры данных.

Ниже описаны стандартные структуры данных, используемые некоторыми стандартными системными вызовами.

8.4.3.1. Стандартные параметры.

ParameterBlock:

Имя	Тип
Name	NameBlock
Value	ValueBlock

Name: имя параметра.
Value: значение параметра ExtendedParameterBlock:

Имя	Тип
Flags	32-битовое битовое поле
Parameter	ParameterBlock

Flags: вектор логических флагов.
Parameter: блок параметра содержащий имя и значение.
NameBlock:

Имя	Тип
Size	32-битовое целое
Characters	Массив 8-битовых символов

Size: 32-битовое беззнаковое целое, равное размеру в байтах следующего за ним поля "characters". Если это значение равно 0, поле characters остается пустым (т.е. после него ничего нет).
Characters: строка UTF-8, оканчивающая символом конца строки.
ValueBlock:

Имя	Тип
Type	32-битовое целое
Size	32-битовое целое
Data	Массив 8-битовых байтов

Type: 32-битовый идентификатор типа. В одном варианте осуществления, заданы следующие типы:

Идентификатор	Имя типа	Описание
0	Integer	32-битовое целочисленное значение, закодированное в виде четырех 8-битовых байтов в обратном порядке следования байтов. В одном варианте осуществления значение считается знаковым, если не указано обратное.
1	Real	32-битовое значение с плавающей точкой, закодированное согласно IEEE-754 в обратном порядке следования байтов
2	String	Строка UTF-8, оканчивающая символом конца строки

3	Date	32-битовое беззнаковое целочисленное значение, представляющее число минут, прошедших с 00:00:00 1 января 1970 г. В одном варианте осуществления, если не указано обратное, значение считается датой UTC, старший бит которого должен быть равен 0.
4	Parameter	Структура ParameterBlock
5	ExtendedParameter	Структура ExtendedParameterBlock
6	Resource	Значение является ресурсом. Ссылка на ресурс осуществляется по ID: поле Data значения представляет собой строку ASCII, оканчивающуюся символом конца строки, содержащую ID ресурса, ссылку на который нужно снять, чтобы получить фактические данные.
7	ValueList	Массив значения (кодированный как ValueListBlock)
8	ByteArray	Значение представляет собой массив 8-битовых байтов

Size: 32-битовое беззнаковое целое, равное размеру в байтах следующего за ним поля "data". Если это значение равно 0, поле данных остается пустым (т.е. после поля size в ValueBlock ничего нет).

Data: массив 8-битовых байтов, представляющий значение. Фактические байты зависят от кодировки данных, указанной в поле type.

ValueListBlock:

Имя	Тип
ValueCount	32-битовое целое
Value0	ValueBlock
Value1	ValueBlock
...	...

ValueCount: 32-битовое беззнаковое целое, равное количеству следующих после него структур ValueBlock. Если это значение равно 0, структур ValueBlock нет.

Value0, Value1, ...: последовательность из нуля или более структур ValueBlock.

8.4.3.2. Стандартная структура ExtendedStatus.

Стандартная структура ExtendedStatusBlock это структура данных, обычно используемая для переноса расширенной информации в качестве состояния возврата из вызова процедуры или системного вызова. Это общего вида структура данных, которую можно использовать в различных контекстах, причем ее поля могут принимать разнообразные значения. В одном варианте осуществления, ExtendedStatusBlock задается следующим образом.

ExtendedStatusBlock:

Имя	Тип
GlobalFlags	32-битовое битовое поле
Category	32-битовое целое
SubCategory	32-битовое целое
LocalFlags	32-битовое битовое поле
CacheDuration	CacheDurationBlock
Parameters	ValueListBlock

GlobalFlags: логические флаги, имеющие одну и ту же семантику вне зависимости от поля Category. Позиция и смысл флагов определяются профилями, которые используют стандартные структуры данных ExtendedStatusBlock.

Category: Уникальный целочисленный идентификатор категории, к которой принадлежит это состояние. Значения идентификатора категории определяются профилями, которые используют стандартные структуры данных ExtendedStatusBlock.

SubCategory: Целочисленный идентификатор (уникальный в рамках категории) подкатегории, которая дополнительно классифицирует тип состояния, описанного этим блоком.

LocalFlags: Логические флаги, семантика которых локальна по отношению к категории и подкатегории этого блока состояний. Позиция и смысл флагов определяются профилями, которые задают и используют семантику категории.

CacheDuration: Указывает продолжительность времени, в течение которого это состояние может кэшироваться (т.е. остается действительным). См. ниже определение типа CacheDurationBlock, чтобы понять, как задается фактическое значение продолжительности времени.

Parameters: список из нуля или более структур ValueBlock. Каждая структура ValueBlock содержит параметр, кодированный как значение типа Parameter или ExtendedParameter. Каждый параметр привязывает имя к типизированному значению и используется для кодирования гибких изменяемых данных, которые описывают блок состояний более подробно, чем просто категория, подкатегория, время жизни кэша и флаги.

CacheDurationBlock:

Имя	Тип
Type	32-битовое целое
Value	32-битовое целое

Type: Целочисленный идентификатор типа значения.

В одном варианте осуществления, заданы следующие типы:

Тип	Описание
0	Значение представляет собой 32-битовое беззнаковое целое, которое представляет число секунд от текущего времени. Значение 0 означает, что состояние вовсе нельзя кэшировать и потому можно использовать только один раз. Особое значение 0xFFFFFFFF интерпретируется как бесконечная длительность (т.е., состояние можно кэшировать сколь угодно долго).
1	Значение представляет собой 32-битовое беззнаковое целое, которое представляет абсолютное местное время, выраженное числом минут, прошедших с 00:00:00 1 января 1970 г. В одном варианте осуществления, старший бит должен быть равен 0.

Value: 32-битовое целое, смысл которого зависит от поля Type.

8.4.4. Стандартные коды результата.

Стандартные коды результата используются в различных API. Для использования в более специальных API можно задать другие коды результата.

Значение	Имя	Описание
0	SUCCESS	Успех
-1	FAILURE	Неудача по неустановленной причине
-2	ERROR_INTERNAL	Произошла внутренняя ошибка (связанная с реализацией)
-3	ERROR_INVALID_PARAMETER	Параметр имеет недействительное значение
-4	ERROR_OUT_OF_MEMORY	Недостаточно памяти для успешного завершения
-5	ERROR_OUT_OF_RESOURCES	Недостаточно ресурсов для успешного завершения
-6	ERROR_NO_SUCH_ITEM	Запрошенный элемент не существует или не найден
-7	ERROR_INSUFFICIENT_SPACE	Вызывающая сущность выделила недостаточно памяти (обычно используется, когда буфер возврата слишком мал)
-8	ERROR_PERMISSION_DENIED	Разрешение на выполнение вызова отменено вызывающей сущностью.
-9	ERROR_RUNTIME_EXCEPTION	Возникла ошибка при выполнении байт-кода
-10	ERROR_INVALID_FORMAT	Ошибка, обусловленная недействительным форматом данных (например, недействительные данные в кодовом модуле)

8.5. Синтаксис ассемблера.

В этом разделе описан иллюстративный синтаксис для использования при компилировании программы в формат байт-кода описанный здесь в другом месте. Очевидно, что это лишь один пример возможного синтаксиса, и что можно использовать любой подходящий синтаксис. Как указано выше, следует также понимать, что представленный здесь формат байт-кода также является иллюстративным, и что описанные здесь системы и способы можно использовать с любым другим пригодным форматом байт-кода или другим форматом кода.

Ассемблер считывает исходные файлы, содержащие код, данные и команды обработки, и создает двоичные кодовые модули, которые могут загружаться виртуальной машиной управления. В одном иллюстративном варианте осуществления, ассемблер обрабатывает исходный файл последовательно, строку за строкой. Строки могут содержать нуль или более символов, после которых следует разделитель строк. Каждая строка может представлять собой: пустую строку (только пробел), директиву сегмента, директиву данных, директиву ассемблера, команду кода, метку или директиву экспорта. Кроме того, каждая строка может заканчиваться комментарием, который начинается с символа ';' и продолжается до конца строки.

Данные и команды, считанные из исходных файлов, имеют неявный сегмент назначения (т.е., куда они попадут, будучи загружены VM). В любой момент в ходе процесса анализа, ассемблер будет иметь "текущий" сегмент, который является неявным сегментом назначения для данных и команд. Текущий сегмент можно менять с использованием директив сегмента.

8.5.1. Директива сегмента.

Директивы сегмента изменяют текущий сегмент анализатора. В одном варианте осуществления поддерживаются директивы сегмента .code и .data. Сегмент .code содержит команды байт-кода, и сегмент .data содержит глобальные переменные.

8.5.2. Директивы данных.

Директивы данных указывают данные (например, целые числа и строки), которые будут загружены в сегмент данных виртуальной машины. В одном варианте осуществления поддерживаются следующие директивы данных:

.string "<some chars>" - Задает строку символов. В одном варианте осуществления ассемблер добавляет октет со значением 0 в конец строки.

.byte <value> - Задает 8-битовое значение. <value> может выражаться десятеричным числом или шестнадцатеричным числом (предваряемым 0x).

8.5.3. Директивы ассемблера.

В одном варианте осуществления поддерживается директива ассемблера .equ <symbol>, <value>, которая задает символ <symbol> равным значению <value>. Символы обычно используются в качестве операндов или кодовых команд.

8.5.4. Labels.

Метки это символы, указывающие те или иные позиции в сегментах. Метки, указывающие на команды в сегменте кода, обычно используются для команд перехода/ветвления. Метки, указывающие на данные в сегменте данных, обычно используются для ссылки на переменные. В одном варианте осуществления используется следующий синтаксис метки: <LABEL>:

Заметим, что после ":" ничего не стоит, кроме необязательного комментария. Метка указывает позицию следующего(ей) элемента данных или команды. В одном варианте осуществления разрешено иметь более одной метки, указывающей один и тот же адрес.

8.5.5. Директивы экспорта.

Директивы экспорта используются для создания элементов в секции "export" кодового модуля, создаваемого ассемблером. Каждый элемент в секции экспорта представляет собой пару (имя, адрес). В рассматриваемом иллюстративном варианте осуществления в секции экспорта можно указывать только адреса в сегменте кода.

Синтаксис директивы экспорта: .export <label>, которая экспортирует адрес, указанный посредством <label>, под именем "<label>".

8.5.6. Кодовые команды.

При компилировании данных, предназначенных для сегмента кода, ассемблер считывает команды, которые отображаются, прямо или косвенно, в байт-коды. В иллюстративном наборе команд, показанном выше, большинство байт-кодов виртуальной машины не имеют прямых операндов и выглядят как простой мнемокод в одной строке. Чтобы сделать синтаксис ассемблера более читаемым, некоторые команды принимают псевдооперанды, которые выглядят так, как если бы они были операндами байт-кода, но в действительности таковыми не являются; в этом случае, ассемблер генерирует одну или несколько команд байт-кода для создания такого же эффекта, как если бы команда имела прямой операнд. Например, команды ветвления используют псевдооперанды.

8.5.6.1. Операнды ветвления.

Команды ветвления можно задавать дословно (безо всяких операндов) или с необязательным операндом, который ассемблер будет преобразовывать в соответствующую последовательность байт-кодов. Необязательный операнд является целочисленной константой или символом. Когда операнд является символом, ассемблер вычисляет правильное целочисленное относительное смещение, чтобы ветвление заканчивалось по адресу, соответствующему символу.

8.5.6.2. Операнды Push.

В одном варианте осуществления команда PUSH всегда берет один операнд. Операнд может представлять собой целочисленную константу, символ или префикс "@", после которого сразу следует имя метки. Когда операнд является символом, проталкиваемое значение является непосредственным значением этого символа, независимо от того, является ли символ меткой или символом .equ (значение не увеличивается на величину смещения сегмента). Когда операнд является именем метки, предваряемым "@", проталкиваемое значение зависит от того, на что указывает метка. Значение, проталкиваемое в стек, является абсолютным адресом, представленным меткой (т.е. суммой локального значения метки и смещения сегмента).

8.5.7. Примеры.

```
; константы
.equ SOMECONST, 7

; которые следуют, идут в сегмент данных
```

```
.data
VAR1:
.byte 8
VAR2:
.string "hello\0"
VAR3:
.long 0xFFFCDA07
VAR4:
.long 0

; которые следуют, идут сегмент кода
.code
FOO:
PUSH 1
ADD
RET
BAR:
PUSH 2
      PUSH @FOO      ; протолкнуть адрес метки FOO
JSR             ; перейти к коду по метке FOO
PUSH SOMECONST ; протолкнуть значение 7
PUSH @VAR1     ; протолкнуть адрес VAR1
PUSH VAR1      ; протолкнуть смещение VAR1 в сегменте данных
PUSH @VAR3     ; протолкнуть адрес VAR3
PEEK           ; протолкнуть значение VAR3
      PUSH @VAR4     ; протолкнуть адрес VAR4
POKE           ; сохранить значение на вершине стека в VAR4
PUSH @VAR2     ; протолкнуть адрес строки "hello"
```

8.5.8. Синтаксис командной строки.

В одном варианте осуществления, ассемблер представляет собой инструмент командной строки, который можно вызывать с помощью следующего синтаксиса:

"PktAssembler[опции]<input_file_path><output_file_path>", где [опции] могут быть: -cs int, -ds int, -xml id или -h, где "-cs int." это Сегмент кода Address Value (default = 8), "-ds int" представляет собой значение адреса сегмента данных (по умолчанию = 4), "-xml id" используется для вывода объекта управления в виде файла XML с указанным ID, и "-h" используется для отображения информации помощи.

9. Объекты управления.

В этом разделе описаны иллюстративные варианты осуществления объектов управления. Объекты управления можно использовать для представления правил, которые регламентируют доступ к контенту путем разрешения или запрещения использования объектов ContentKeys, которыми они управляют. Их также можно использовать для представления ограничений по действительности объекта «связь», в который они внедрены. Их также можно использовать в качестве самостоятельных программных контейнеров, которые выполняются от имени другой сущности, например, в агентах или делегатах. В одном варианте осуществления объекты управления содержат метаданные и программы в виде байт-кода, которые реализуют конкретный протокол взаимодействия. Протокол управления имеет целью задавать взаимодействие между механизмом DRM и программой управления или между приложением хоста и программой управления через механизм DRM. В этом разделе также описаны иллюстративные действия, которые приложение может осуществлять на контенте, параметры действия, которые нужно передавать программе управления, и показано, как программа управления кодирует состояние возврата, указывающее, что запрошенное действие может или не может быть выполнено, а также параметры, которые могут дополнительно описывать состояние возврата.

В этом разделе используются следующие аббревиатуры и акронимы:

ESB: расширенный блок состояний;

LSB: младший бит;

Byte: 8-битовое значение или октет;

Байт-код: поток байтов, которые кодируют исполнимые команды и их операнды.

9.1. Программы управления.

В одном варианте осуществления объект управления содержит программу управления. Программа управления включает в себя кодовый модуль, содержащий байт-код, который может выполнять виртуальная машина, и список именованных процедур (например, элементов таблицы экспорта).

В одном варианте осуществления набор процедур, которые представляют правила, регламентирующие осуществление определенной операции (например "воспроизведение") на элементе контента, называется 'управляющий элемент действия'. Набор процедур, которые представляют ограничения по действительности на объекте «связь», называется "ограничение по связи". Набор процедур, предназначенных для выполнения от имени удаленной сущности (например, в течение сеанса протокола, когда механизм DRM выполняется на другом хосте), называется "агент". Набор процедур, предназначенных для выполнения от имени другого управляющего элемента (например, когда программа управления использует системный вызов System.Host.CallVm), называется "делегат".

9.1.1. Интерфейс к программам управления.

В одном варианте осуществления программы управления выполняются виртуальной машиной, выполняющейся в среде хоста. Среду хоста можно реализовать любым подходящим способом; однако, для простоты объяснения и в целях иллюстрации, в нижеследующем рассмотрении предполагается, что реализацию среды хоста виртуальной машины можно логически разделить на две части: приложение хоста и механизм DRM. Однако очевидно, что другие варианты осуществления могут иметь другое логическое разделение функций, которое может быть эквивалентным вышеописанной логической структуре.

Как и на фиг. 29, в предпочтительных вариантах осуществления механизм DRM 2908 является логическим интерфейсом между приложением хоста 2900 и программами управления 2906. Приложение хоста 2900 делает логические запросы механизму 2908, например, запрашивая доступ к ключу контента с определенной целью (например, для воспроизведения или представления потока контента). В одном варианте осуществления механизм 2908 гарантирует, что описанный ниже протокол взаимодействия реализован правильно, например, гарантируя, что любые гарантии, касающиеся инициализации программы управления, последовательности вызовов и других деталей взаимодействия соблюдаются.

Когда приложение хоста 2900 запрашивает использование ключей контента для набора ID контента, механизм DRM 2908 определяет, какой объект Control использовать. Объекты Protector позволяют механизму определять, к каким объектам ContentKey нужно обращаться на предмет запрошенных ID контента. Затем механизм находит объект Controller, который ссылается на эти объекты ContentKey. В одном варианте осуществления, объект Controller может ссылаться на более чем один объект ContentKey.

Это позволяет управлять несколькими объектами ContentKey посредством одного и того же объекта Control. Когда приложение хоста запрашивает доступ к ключу контента путем вызова действия, оно может запросить множественные ID контента как группу, в той мере, в какой один и тот же объект Controller ссылается на соответствующие им объекты ContentKey. В одном варианте осуществления, запрос на доступ к группе ключей контента, на которые ссылаются более одного объекта «контроллер», не разрешен.

В одном варианте осуществления механизм DRM следует соглашению по отображению действий в имена процедур. Например, в одном варианте осуществления для каждой из описанных ниже процедур, имя, которое появляется в элементе таблицы экспорта в кодовом модуле, является соответствующей строкой, показанной ниже в разделах 9.1.4-9.1.7.

9.1.1.1. Загрузка объекта управления.

В одном варианте осуществления прежде чем механизм сможет произвести вызовы процедур управления, ему необходимо загрузить кодовый модуль объекта управления в виртуальную машину. В одном варианте осуществления в одну VM загружается только один кодовый модуль.

9.1.1.2. Атомарность.

В одном варианте осуществления механизм гарантирует, что вызовы процедур в программах управления являются атомарными в отношении ресурсов, выделяемых процедуре, например, базы данных объектов (или "состояний"). Таким образом, в этом варианте осуществления, механизм должен гарантировать, что эти ресурсы остаются неизменными в ходе выполнения любой процедуры, которую он вызывает. Для этого можно эффективно блокировать эти ресурсы при вызове процедуры, или можно запретить одновременное выполнение нескольких VM. Однако механизму не нужно гарантировать, что эти ресурсы останутся неизменными при последующих вызовах процедуры.

9.1.2. Протокол управления.

В одном варианте осуществления, именование процедур, входной/выходной интерфейс и структуры данных для каждой процедуры в кодовом модуле, вместе образуют протокол управления. Протокол, реализованный кодовым модулем, указан в поле "protocol" объекта управления. Описанный ниже иллюстративный протокол управления будем называть Стандартным протоколом управления, и его идентификатор (значение поля 'protocol') есть "http://www.octopus-drm.com/specs/scp-l_0".

В одном варианте осуществления прежде чем механизм DRM загрузит кодовый модуль и вызовет процедуры в программе управления, ему нужно гарантировать, что взаимодействие с программой управления будет соответствовать спецификации конкретного id протокола, указанного в поле «протокол». Это включает в себя любую гарантию относительно особенностей виртуальной машины, которые необходимо реализовать, гарантии относительно размера пространства адресов, доступного программе управления, и т.п.

Протоколы управления, например Стандартный протокол управления, можно усовершенствовать со временем без необходимости создавать новую спецификацию протокола. Пока изменения, вносимые в протокол, согласуются с предыдущими ревизиями спецификации, и пока существующие реализации механизма DRM, а также существующие программы управления, которые согласуются с этим протоколом, продолжают осуществляться согласно спецификации, изменения считаются совместимыми. Такие изменения могут включать в себя, например, новые типы действий.

9.1.3. Тип байт-кода.

В вышеописанном иллюстративном варианте осуществления, предусматривающем Стандартный протокол управления, тип байт-кодового модуля это "Байт-кодовый модуль Plankton версия 1.0". В этом иллюстративном варианте осуществления, значение поля "type" объекта управления есть

"http://www.octopus-drm. com/specs/pkcm-1_0".

9.1.4. Общие процедуры управления.

Общие процедуры - это процедуры, которые применимы для объектов управления в целом, и не предназначены для данного действия или ограничения по связи. В одном иллюстративном варианте осуществления используются следующие общие процедуры управления.

9.1.4.1. Control.Init

Эта процедура является необязательной (т.е. она не требуется во всех объектах управления). Если эта процедура используется, механизм вызывает ее один раз до вызова любой другой процедуры управления. Процедура не имеет аргументов и возвращает ResultCode в вершину стека в качестве выхода. ResultCode равен 0 в случае успеха или отрицательному коду ошибки в случае неудачи. В одном варианте осуществления, если ResultCode не равен 0, механизм прерывает текущую операцию объекта управления и не делает никаких дополнительных вызовов процедур для этого объекта управления.

9.1.4.2. Control.Describe

Эта процедура является необязательной. Процедура вызывается, когда приложение запрашивает описание смысла правил, представленных программой управления в целом (т.е. не для конкретного действия). Процедура не имеет аргументов и возвращает ResultCode и StatusBlockPointer в вершину стека в качестве выходов, где ResultCode является целочисленным значением (0 если процедура завершена успешно, или, в противном случае, отрицательный код ошибки), и где StatusBlockPointer это адрес стандартного ExtendedStatusBlock. ExtendedStatusBlock содержит информацию, которую приложение может интерпретировать и использовать для предоставления пользователю информации относительно смысла правил, представленных программой управления.

9.1.4.3. Control.Release

Эта процедура является необязательной. Если эта процедура существует, механизм DRM вызывает ее один раз после того, как ему уже не нужно вызывать какую-либо другую процедуру для объекта управления. Никакая другая процедура не будет вызвана для объекта управления, пока не будет инициировано новое использование объекта управления (в каковом случае, снова будет вызвана процедура Control.Init). Процедура не имеет аргументов и возвращает ResultCode в вершину стека в качестве выхода.

ResultCode равен 0 в случае успеха или отрицательному коду ошибки в случае неудачи.

9.1.5. Процедуры Action

Всякое возможное действие имеет имя (например, «воспроизведение», «перенос», «экспорт» и т.д.). В одном иллюстративном варианте осуществления, для данного действия <Action>, заданы следующие имена процедур (где "<Action>" обозначает фактическое имя действия (например, "play", "transfer", "export", и т.д.)).

9.1.5.1. Control.Actions.<Action>.Init

Эта процедура является необязательной. Если она существует, механизм вызывает ее один раз до того, как вызвать какую-либо другую процедуру для этого действия. Процедура не имеет аргументов и возвращает ResultCode в вершину стека в качестве выхода. ResultCode равен 0 в случае успеха или отрицательному коду ошибки в случае неудачи. В одном варианте осуществления, если ResultCode не равен 0, механизм прерывает текущее действие и не делает никаких дополнительных вызовов процедур для этого действия в этом объекте управления.

9.1.5.2. Control.Actions.<Action>.Check

В рассматриваемом иллюстративном варианте осуществления, эта процедура является обязательной и вызывается для проверки, без фактического осуществления данного действия, каково было бы состояние возврата, если бы для этого действия была вызвана процедура Perform. Важно, чтобы эта процедура не имела побочных эффектов. Заметим, что если процедура Perform также не имеет побочных эффектов, элементы Check и Perform в таблице элементов объекта управления могут указывать на ту же процедуру. Эта процедура имеет те же входы и выходы, что и описанная ниже процедура Perform.

9.1.5.3. Control.Actions.<Action>.Perform

В одном варианте осуществления эта процедура является обязательной и вызывается, когда приложение собирается осуществить действие. Процедура не имеет аргументов и возвращает ResultCode и StatusBlockPointer в вершину стека в качестве выходов, где ResultCode это целочисленное значение (0 если процедура завершена успешно, или, в противном случае, отрицательный код ошибки), и где StatusBlockPointer это адрес стандартного ExtendedStatusBlock. Заметим, что в одном варианте осуществления ResultCode успеха (т.е. 0) не означает, что запрос удовлетворен. Он означает только то, что процедура выполнена без ошибок. Удовлетворен или отклонен запрос, указывает ExtendedStatusBlock. Однако, если ResultCode указывает неудачу, приложение хоста реагирует, как будто запрос отклонен. Например, в одном варианте осуществления категория структуры StatusBlock должна быть ACTION_DENIED, или возвращаемый ExtendedStatusBlock должен отвергаться, и тогда приложение хоста прерывает действие.

При осуществлении действия нужно вызывать только процедуру Perform. Механизму не нужно предварительно вызывать процедуру Check. Реализация процедуры Perform может предусматривать внутренний вызов процедуры Check, по ее собственному усмотрению, но не следует полагать, что система будет заранее вызывать процедуру Check.

9.1.5.4. Control.Actions.<Action>.Describe

Эта процедура является необязательной и вызывается, когда приложение запрашивает описание смысла правил и условий, представленных программой управления для данного действия. Процедура не имеет аргументов и возвращает ResultCode и StatusBlockPointer в вершину стека в качестве выходов, где ResultCode это целочисленное значение (0 если процедура завершена успешно, или, в противном случае, отрицательный код ошибки), и где StatusBlockPointer это адрес стандартного ExtendedStatusBlock.

9.1.5.5. Control.Actions.<Action>.Release

Эта процедура является необязательной. Если она существует, она вызывается один раз, когда механизму DRM уже не нужно вызывать какие-либо другие процедуры для данного действия. Для данного действия не вызываются никакие другие процедуры, пока не будет инициировано новое использование действия (в каковом случае, снова будет вызвана процедура Init). Процедура не имеет аргументов и возвращает ResultCode в вершину стека в качестве выхода. ResultCode равен 0 в случае успеха и отрицательному коду ошибки в случае неудачи. Если ResultCode не равен 0, механизм не делает никаких дополнительных вызовов процедур для данного действия.

9.1.6. Процедуры ограничения по связи.

В одном варианте осуществления, когда в объект «связь» внедрен объект управления, механизм DRM вызывает процедуры ограничения по связи в этом объекте управления для проверки действительности объекта «связь». В одном иллюстративном варианте осуществления используются следующие процедуры ограничения по связи.

9.1.6.1. Control.Link.Constraint.Init

Эта процедура является необязательной, и, если существует, вызывается только один раз прежде, чем будет вызвана какая-либо другая процедура для данного ограничения по связи. Процедура не имеет аргументов и возвращает ResultCode в вершину стека в качестве выхода. ResultCode равен 0 в случае успеха и отрицательному коду ошибки в случае неудачи. Если ResultCode не равен 0, механизм считает, что ограничение по действительности для объекта «связь» не выполнено, и блокирует дальнейшие вызовы процедур для объекта управления связи.

9.1.6.2. Control.Link.Constraint.Check

В рассматриваемом иллюстративном варианте осуществления, эта процедура является обязательной и вызывается для проверки, выполняется ли ограничение по действительности для данной связи. Процедура не имеет аргументов и возвращает ResultCode и StatusBlockPointer в вершину стека в качестве выходов, где ResultCode это целочисленное значение (0 если процедура завершена успешно, или, в противном случае, отрицательный код ошибки), и где StatusBlockPointer это адрес стандартного ExtendedStatusBlock. Если ResultCode не равен 0, механизм считает, что ограничение по действительности для объекта «связь» не выполнено, и блокирует дальнейшие вызовы процедур для объекта управления связи. Даже если ResultCode равен 0 (успех), это не означает, что ограничение выполнено; это означает только то, что процедура выполнена без ошибок.

Выполнено ограничение или нет, указывает StatusBlock.

9.1.6.3. Control.Link.Constraint.Describe

Эта процедура является необязательной и вызывается, когда приложение запрашивает описание смысла ограничения, представленного программой управления для данной связи. Процедура не имеет аргументов и возвращает ResultCode и StatusBlockPointer в вершину стека в качестве выходов, где ResultCode это целочисленное значение (0 если процедура завершена успешно, или, в противном случае, отрицательный код ошибки), и где StatusBlockPointer это адрес стандартного ExtendedStatusBlock.

9.1.6.4. Control.Link.Constraint.Release

Эта процедура является необязательной и, если существует, вызывается механизмом после того, как механизму уже не нужно вызывать какую-либо другую процедуру для данного ограничения. Процедура не имеет аргументов и возвращает ResultCode в вершину стека в качестве выхода. ResultCode равен 0 в случае успеха и отрицательному коду ошибки в случае неудачи. Согласно рассматриваемому варианту осуществления, после вызова этой процедуры, нельзя вызвать никакую другую процедуру для данного ограничения, пока не будет инициирован новый цикл (в каковом случае, снова вызвается процедура Init). Аналогично, если ResultCode не равен 0, механизм не делает никаких дополнительных вызовов процедур для данного ограничения по связи.

9.1.7. Процедуры агента.

В одном варианте осуществления агент это объект управления, который призван выполняться от имени сущности. Агенты обычно используются в контексте взаимодействия услуг между двумя конечными точками, когда одна конечная точка должна выполнять код некоторой виртуальной машины в контексте второй конечной точки и, возможно, получать результат этого выполнения. В одном варианте осуществления объект управления может содержать несколько агентов, и каждый агент может содержать любое количество процедур, которые могут выполняться; однако, на практике, агенты обычно имеют по одной процедуре.

В одном иллюстративном варианте осуществления заданы следующие точки входа для агентов, где <Agent> это строка имени, которая является ссылкой на фактическое имя агента.

9.1.7.1. Control.Agents.<Agent>.Init

Эта процедура является необязательной и, если она существует, механизм вызывает ее один раз до того, как какая-либо другая процедура будет вызвана для данного агента. Процедура не имеет аргументов и возвращает ResultCode в вершину стека в качестве выхода. ResultCode равен 0 в случае успеха и отрицательному коду ошибки в случае неудачи.

9.1.7.2. Control.Agents.<Agent>.Run

В рассматриваемом иллюстративном варианте осуществления, эта процедура является обязательной и является главной процедурой агента. Процедура не имеет аргументов и возвращает ResultCode, ReturnBlockAddress и ReturnBlockSize в вершину стека в качестве выходов. ResultCode это целочисленное значение (0 если процедура завершена успешно, или, в противном случае, отрицательный код ошибки), ReturnBlockAddress это адрес блока памяти, предназначенного для хранения данных, которые агентский код должен возвратить вызывающей сущности (если процедура ничего не должна возвращать, адрес равен 0), и ReturnBlockSize это размер в байтах блока памяти по адресу ReturnBlockAddress. В одном варианте осуществления, если ReturnBlockAddress равен 0, то значение ReturnBlockSize также равно 0.

9.1.7.3. Control.Agents.<Agent>.Describe

Эта процедура является необязательной и вызывается, когда приложение запрашивает описание данного агента. Процедура не имеет аргументов и возвращает ResultCode и StatusBlockPointer в вершину стека в качестве выходов, где ResultCode это целочисленное значение (0 если процедура завершена успешно, или, в противном случае, отрицательный код ошибки), и где StatusBlockPointer это адрес стандартного ExtendedStatusBlock.

9.1.7.4. Control.Agents.<Agent>.Release

Эта процедура является необязательной и, если она существует, механизм вызывает ее один раз, когда механизму уже не нужно вызывать какие-либо другие процедуры для этого агента.

Для этого агента не будет вызвана никакая другая процедура, пока не будет инициирован новый цикл (в каковом случае, снова будет вызвана процедура Init). Процедура не имеет аргументов и возвращает ResultCode в вершину стека в качестве выхода. ResultCode равен 0 в случае успеха и отрицательному коду ошибки в случае неудачи.

9.2. Расширенные блоки состояний.

Нижеследующие иллюстративные определения применимы к структурам данных ExtendedStatusBlock, возвращаемым несколькими вышеописанными процедурами согласно иллюстративным вариантам осуществления. Примеры структур данных ExtendedStatusBlock описаны в связи с описанием виртуальной машины.

В одном варианте осуществления, не существует глобальных флагов для ExtendedStatusBlock. В этом варианте осуществления, программы управления задают поле GlobalFlag структуры ExtendedStatuBlock равным 0.

9.2.1. Категории.

В нижеследующих разделах заданы значения для поля Category структур ExtendedStatusBlock согласно одному варианту осуществления. В одном варианте осуществления, ни одна из этих категорий не имеет подкатегорий, и поэтому значение поля SubCategory структур ExtendedStatusBlock задано равным 0.

В одном варианте осуществления заданы следующие коды категорий.

9.2.1.1. Процедуры Check и Perform для действий.

Значение	Имя	Описание
0	ACTION_GRANTED	Приложение авторизовано использовать ключи контента, которыми управляет программа управления, для выполнения запрошенного действия. Список параметров возвращаемой структуры ExtendedStatusBlock не должен содержать никаких параметров ограничения, но может

		содержать параметры обязательства и/или обратного вызова.
1	ACTION_DENIED	Приложение не авторизовано использовать ключи контента, которыми управляет программа управления, для выполнения запрошенного действия. Когда действие отменено, программа управления должна включить в список параметров возвращаемой структуры ExtendedStatusBlock одно или несколько ограничений, которые не были выполнены, что привело к отмене действия (ограничения, которые не оценивались, и ограничения, которые не привели к отмене действия следует пропустить). В одном варианте осуществления, список параметров возвращаемой структуры ExtendedStatusBlock не должен содержать никаких параметров обязательства или обратного вызова.

В одном варианте осуществления в контексте параметров ExtendedStatusBlock, возвращаемых процедурами действия, ограничение означает условие, которое должно быть выполнено, или критерий, которому нужно удовлетворить, чтобы процедура возвратила ExtendedStatusBlock с категорией ACTION_GRANTED.

В одном варианте осуществления значения поля LocalFlags, общие для обеих вышеописанных категорий, включают в себя:

Битовый индекс (0 это LSB)	Имя	Описание
0	OBLIGATION_NOTICE	Список параметров содержит один или несколько параметров, которые относятся к обязательствам
1	CALLBACK_NOTICE	Список параметров содержит один или несколько параметров, которые относятся к обратным вызовам
2	GENERIC_CONSTRAINT	Список параметров содержит один или несколько параметров, которые относятся к общим ограничениям
3	TEMPORAL_CONSTRAINT	Список параметров содержит один или несколько параметров, которые относятся к временным ограничениям
4	SPATIAL_CONSTRAINT	Список параметров содержит один или несколько параметров, которые относятся к пространственным ограничениям
5	GROUP_CONSTRAINT	Список параметров содержит один или несколько параметров, которые относятся к групповым ограничениям
6	DEVICE_CONSTRAINT	Список параметров содержит один или несколько параметров, которые относятся к ограничениям по устройству

7	COUNTER_CONSTRAINT	Список параметров содержит один или несколько параметров, которые относятся к ограничениям по счетчику

Согласно вышеприведенной таблице, упомянутый в ней список параметров представляет собой поле "Parameters" структуры данных ExtendedStatusBlock.

9.2.1.2. Коды категорий процедур Describe.

В одном варианте осуществления для процедур Describe коды категорий не заданы. В одном варианте осуществления, к процедурам Describe применяются те же локальные флаги, которые заданы для процедур Action, и процедуры Describe должны включать в возвращаемую ими структуру ExtendedStatusBlock параметр по имени 'Description', описанный ниже. В одном варианте осуществления, процедуры Describe не содержат в возвращаемой ими структуре ExtendedStatusBlock никаких параметров обязательства или обратного вызова; однако процедуры Describe должны включать в возвращаемую ими структуру ExtendedStatusBlock параметры, описывающие некоторые или все ограничения, применимые к соответствующему действию или ограничению по связи.

9.2.1.3. Коды категорий процедур для ограничения по связи.

Значение	Имя	Описание
0	LINK_VALID	Связь, ограниченная этой программой управления, действительна. Список параметров возвращаемой ESB не должен содержать никаких параметров ограничения, и, в одном варианте осуществления, не должен содержать параметров обязательства или обратного вызова
1	LINK_INVALID	Связь, ограниченная этой программой управления недействительна. Когда связь недействительна, программа управления должна включить в список параметров возвращаемой ESB одно или несколько ограничений, которые не были выполнены и привели к недействительности связи (ограничения, которые не оценивались, и ограничения, которые не привели к отмене действия, следует пропустить). В одном варианте осуществления, список параметров возвращаемой ESB не должен содержать никаких параметров обязательства или обратного вызова.

В одном варианте осуществления, для каждой из этих категорий применяются те же локальные флаги, которые заданы для процедур Action.

В одном варианте осуществления, в контексте параметров ExtendedStatusBlock, возвращаемых процедурами для ограничения по связи, ограничение означает условие, которое должно быть выполнено, или критерий, которому нужно удовлетворить, чтобы процедура возвратила ExtendedStatusBlock с категорией LINK_VALID.

9.2.2. Времена жизни кэша.

Поле CacheDuration структуры ExtendedStatusBlock указывает период действия информации, закодированной в ExtendedStatusBlock. Когда ExtendedStatusBlock имеет ненулевой период действия, это оз-

начает, что ExtendedStatusBlock можно сохранять в кэше, и что, в течение этого периода времени, точно такой же процедуры с теми же самыми параметрами возвратит точно такую же структуру ExtendedStatusBlock, поэтому приложению хоста можно возвращать кэшированное значение вместо того, чтобы вызывать процедуру.

9.2.3. Параметры.

Некоторые параметры используются для переноса подробной информации о состоянии возврата, а также связки переменных для обработки шаблонов (см. раздел 9.4).

В одном варианте осуществления, кроме обязательств и обратных вызовов, все описанные здесь ограничения имеют своей единственной целью помощь в классификации и отображении приложения хоста, а не применение правил пользования. За применение правил отвечает программа управления.

В одном варианте осуществления, параметры, заданные в нижеследующем разделе, кодируются либо как ParameterBlock, если никакие флаги параметра не применимы, либо как ExtendedParameterBlock, если один или несколько флагов применимы. Ниже описаны иллюстративные флаги.

9.2.3.1. Описание.

Имя параметра: Description.

Тип параметра: ValueList.

Описание: Список параметров описания. Каждое значение в списке имеет тип Parameter или ExtendedParameter. В одном варианте осуществления, заданы следующие параметры: Default, Short и Long. Каждый из них, если присутствует, имеет в качестве значения ID одного из ресурсов объекта управления. Этот ресурс должен содержать полезную нагрузку в виде текста или полезную нагрузку в виде шаблона. Если ресурс является шаблоном, он обрабатывается для получения текстуального описания результата (описания либо всей программы управления, либо конкретного действия). Шаблон обрабатывается с использованием, в качестве связок переменных, других параметров из списка, в котором присутствует параметр 'Description'.

В одном варианте осуществления, описания 'Short' и 'Long' могут быть включены только, если включено также описание 'Default'.

Имя	Тип	Описание
Default	Resource	Id ресурса, который содержит нормальный текст или шаблон описания
Short	Resource	Id ресурса, который содержит короткий текст или шаблон описания
Long	Resource	Id ресурса, который содержит длинный текст или шаблон описания

9.2.3.2. Ограничения.

В одном варианте осуществления, параметры ограничения сгруппированы в списки, которые содержат ограничения сходных типов. В одном варианте осуществления, заданы стандартные ограничения для некоторых типов. В одном варианте осуществления, объекты управления могут возвращать параметры ограничения, которые не включены в набор стандартных ограничений, при условии, что имя параметра ограничения является URN в пространстве имен, что гарантирует уникальность этого имени. Такие ограничения могут включать в себя ограничения, относящиеся к поставщику, или ограничения, установленные в других спецификациях.

9.2.3.2.1. Общие ограничения.

Имя параметра: GenericConstraints.

Тип параметра: ValueList.

Описание: Список общих ограничений, которые можно применять. Каждое значение в списке имеет тип Parameter или ExtendedParameter.

В одном варианте осуществления, общие ограничения являются ограничениями, которые не принадлежат никакому другому типу ограничения, заданному в этом разделе. В одном варианте осуществления, никакие общие параметры ограничения не заданы.

9.2.3.2.2. Временные ограничения.

Имя параметра: TemporalConstraints.

Тип параметра: ValueList.

Описание: Список временных ограничений, которые можно применять. Каждое значение в списке имеет тип Parameter или ExtendedParameter. Временные ограничения это ограничения, которые относятся к времени, дате, длительности и/или т.п. В одном варианте осуществления заданы следующие временные параметры ограничения:

Имя	Тип	Описание
NotBefore	Date	Дата, до которой действие запрещено
NotAfter	Date	Дата, после которой действие запрещено
NotDuring	ValueList	Список из 2 значений типа Date. Первое значение это начало периода, а второе - конец периода, который исключен
NotLongerThan	Integer	Максимальное число секунд после первого использования. В одном варианте осуществления, это значение является беззнаковым
NotMoreThan	Integer	Максимальное число секунд совокупного времени использования. В одном варианте осуществления, это значение является беззнаковым

9.2.3.2.3. Пространственные ограничения.

Имя параметра: SpatialConstraints.

Тип параметра: ValueList.

Описание. Список пространственных ограничений, которые можно применять. В одном варианте осуществления, каждое значение в списке имеет тип Parameter или ExtendedParameter. Пространственные ограничения это ограничения, которые относятся к физическим положениям. В одном варианте осуществления, никакие стандартные пространственные ограничения не заданы.

9.2.3.2.4. Групповые ограничения.

Имя параметра: GroupConstraints.

Тип параметра: ValueList.

Описание: Список групповых ограничений, которые можно применять. Каждое значение в списке имеет тип Parameter или ExtendedParameter. Групповые ограничения это ограничения, которые относятся к группам, групповой принадлежности, идентификации групп и/или т.п. В одном варианте осуществления заданы следующие параметры:

Имя	Тип	Описание
MembershipRequired	Resource	Id ресурса, который содержит текст или шаблон для имени или идентификатора группы, принадлежность к которой требуется
IdentityRequired	Resource	Id ресурса, который содержит текст или шаблон для имени или идентификатора отдельной сущности

9.2.3.2.5. Ограничения по устройству.

Имя параметра: DeviceConstraints.

Тип параметра: ValueList.

Описание: Список ограничений по устройству, которые можно применять. Каждое значение в списке имеет тип Parameter или ExtendedParameter. Ограничения по устройству это ограничения, которые относятся к характеристикам устройства, например признакам, атрибутам, именам, идентификаторам и/или т.п. В одном варианте осуществления заданы следующие параметры:

Имя	Тип	Описание
DeviceTypeRequired	Resource	Id ресурса, который содержит текст или шаблон для требуемого типа устройства хоста
DeviceFeatureRequired	Resource	Id ресурса, который содержит текст или шаблон для имени признака, который должно иметь устройство хоста
DeviceIdRequired	String	Id, который должно иметь

	устройство. Этот Id может быть любой строкой, которую можно использовать для идентификации устройства (например, именем устройства, серийным номером устройства, id узла и/или т.п.).

9.2.3.2.6. Ограничения по счетчику.

Имя параметра: CounterConstraints.

Тип параметра: ValueList.

Описание: Список ограничений по счетчику, которые можно применять. Каждое значение в списке имеет тип Parameter или ExtendedParameter. Ограничения по счетчику это ограничения, которые относятся к значениям счетчика, например, счетчику воспроизведения, накопленному значению счетчика и/или т.п. В одном варианте осуществления, никакие стандартные ограничения по счетчику не заданы.

9.2.3.3. Флаги параметра.

В одном варианте осуществления, следующие флаги можно использовать для всех параметров, описанных в разделе 9.2.3, когда они закодированы в виде ExtendedStatusBlock:

Битовый индекс (0 это LSB)	Имя	Описание
0	CRITICAL	Приложение хоста должно понимать семантику, связанную с этим параметром. В противном случае, всю структуру ExtendedStatusBlock следует считать непонятной и отвергать. В одном варианте осуществления, этот флаг не следует использовать для параметров, имеющих описательный характер.
1	HUMAN_READABLE	Этот параметр представляет значение, имя и значение которого пригодны для отображения на текстовом или графическом пользовательском интерфейсе. Любой параметр, для которого этот флаг не установлен, следует резервировать для приложения хоста и не следует показывать пользователю. Для значений параметра типа Resource, он не является ID ресурса, но указывает, что полезная нагрузка данных ресурса, указываемая ID, воспринимаема человеком.

9.4. Обязательства и обратные вызовы.

В одном варианте осуществления, определенные действия, когда они разрешены, требуют дополнительного участия приложения хоста. Обязательства представляют операции, которые должно осуществлять приложение хоста после использования ключа контента, которые они запрашивают. Обратные вызовы представляют вызовы одной или нескольких процедур программы управления, которые должно осуществлять приложение хоста после использования ключа контента, которые они запрашивают.

В одном варианте осуществления, если приложение встречает какое-либо критическое обязательство или обратный вызов, которое(ый) оно не поддерживает или не понимает (например, потому, что тип обязательства был задан после реализации приложения), оно должно отказаться продолжать действие, для которого был возвращен этот параметр обязательства или обратного вызова. В одном варианте осуществления, критическое (ий) обязательство или обратный вызов указывается путем задания флага параметра CRITICAL для параметра, который его описывает.

Если управление имеет побочные эффекты (например, уменьшение счетчика воспроизведения), оно должно использовать обратный вызов OnAccept, чтобы потребовать от приложения хоста вызов определенной процедуры, если оно способно понимать все критические обязательства и обратные вызовы и выполнять их. Побочный эффект имеет место в процедуре обратного вызова. В одном иллюстративном

варианте осуществления, реализации должны понимать и реализовывать обратный вызов OnAccept, поскольку это может быть полезно для предотвращения преждевременного возникновения побочных эффектов (например, обновлений базы данных состояний) (например, прежде чем приложение хоста определит, что оно неспособно выполнить данное(ый) критическое обязательство или обратный вызов и должно прекратить выполнение действия), что обеспечивает меру транзакционной атомарности.

9.4.1. Параметры.

Следующие параметры задают несколько типов обязательств и обратных вызовов, которые могут возвращаться в структурах данных ExtendedStatusBlock.

9.4.1.1. Обязательства.

Имя параметра: Obligations.

Тип параметра: ValueList.

Описание: Список параметров обязательства. Каждое значение в списке имеет тип Parameter или ExtendedParameter. В одном варианте осуществления заданы следующие параметры обязательства:

<table>
<tr><th>Имя</th><th>Тип</th><th>Описание</th></tr>
<tr><td>RunAgentOnPeer</td><td>ValueList</td><td>Приложение хоста должно направить объект управления агента для выполнения на равноправном устройстве сеанса протокола, выполняющегося в данный момент.
<table>
<tr><th>Тип</th><th>Описание</th></tr>
<tr><td>String</td><td>Id объекта управления, который содержит агент для выполнения.</td></tr>
<tr><td>String</td><td>Имя агента для выполнения.</td></tr>
<tr><td>Integer</td><td>Id экземпляра. Это значение используется для уникальной идентификации экземпляра обязательство этого агента Этот id также позволяет системе коррелировать обязательство этого агента с параметром обратного вызова OnAgentCompletion.</td></tr>
<tr><td>String</td><td>Id контекста. Этот Id виден агенту, выполняющемуся на равноправном устройстве, по пути контекста сеанса Host Object агента: Octopus/Agent/Parameters/Session/ContextId.</td></tr>
<tr><td>ValueList</td><td>Список значений типа Parameter. Агент видит все эти параметры как входные параметры.</td></tr>
</table></td></tr>
</table>

9.4.1.2. Обратные вызовы.

Имя параметра: Callbacks.

Тип параметра: ValueList.

Описание: список параметров обратного вызова. Каждое значение в списке имеет тип Parameter или ExtendedParameter. В одном варианте осуществления заданы следующие параметры обратных вызовов:

<table>
<tr><th>Имя</th><th>Тип</th><th>Описание</th></tr>
<tr><td>OnAccept</td><td>Callback</td><td>Приложение хоста должно осуществлять обратный вызов, если оно способно понимать все параметры критических обязательств и обратных вызовов, содержащиеся в этой ESB.
В одном варианте осуществления, может существовать, самое большее, один параметр обратного вызова OnAccept в списке параметров обратных вызовов. Если в списке указаны другие параметры обратных вызовов, OnAccept выполняется в первую очередь.</td></tr>
<tr><td>OnTime</td><td>ValueList</td><td>Приложение хоста должно осуществлять обратный вызов после указанной/го даты/времени.<table><tr><th>Тип</th><th>Описание</th></tr><tr><td>Date</td><td>Дата, после которой приложению хоста нужно осуществлять обратный вызов.</td></tr><tr><td>Callback</td><td>Процедура для обратного вызова и соответствующий куки.</td></tr></table></td></tr>
<tr><td>OnTimeElapsed</td><td>ValueList</td><td>Приложение хоста должно осуществлять обратный вызов по истечении указанного срока (отсчет начинается, когда приложение хоста фактически осуществляет действие, на которое дано разрешение).<table><tr><th>Тип</th><th>Описание</th></tr><tr><td>Integer</td><td>Число секунд. Значение является беззнаковым.</td></tr><tr><td>Callback</td><td>Процедура для обратного вызова и соответствующий куки.</td></tr></table></td></tr>
<tr><td>OnEvent</td><td>ValueList</td><td>Приложение хоста должно осуществлять обратный вызов, когда наступает определенное событие.<table><tr><th>Тип</th><th>Описание</th></tr><tr><td>String</td><td>Имя события</td></tr><tr><td>Integer</td><td>Флаги события (целочисленное значение интерпретируется как битовое поле)</td></tr><tr><td>Integer</td><td>Параметр события</td></tr><tr><td>Callback</td><td>Процедура для обратного вызова и соответствующий куки.</td></tr></table>Более подробные сведенья о событиях см. в разделе о событиях.</td></tr>
<tr><td>OnAgentCompletion</td><td>ValueList</td><td>Приложение хоста должно осуществлять обратный вызов, когда выполнение агента,</td></tr>
</table>

указанного в одном из параметров обязательства, завершено или не удалось.

Тип	Описание
Integer	Id экземпляра агента. Id экземпляра, указанный в обязательстве агента.
Callback	Процедура для обратного вызова и соответствующий куки.

При осуществлении обратного вызова, приложение хоста должно обеспечивать следующий ArgumentsBlock:

Тип	Кодировка	Описание
32-битовое целое	4 байта в обратном порядке следования байтов	Код состояния «завершение»
32-битовое целое	4 байта в обратном порядке следования байтов	Код результат агента
Массив 8-битовых байтов	Последовательность байтов	ReturnBlock агента

Значение кода состояния завершения равно 0, если агент выполнен нормально, или отрицательному коду ошибки, если нет.

ReturnBlock агента это данные, возвращаемые агентом. Он отсутствует, если не удалось выполнить агент (код состояния завершения не равен 0).

В одном варианте осуществления тип 'Callback', указанный в вышеприведенной таблице, представлен как ValueListBlock с тремя элементами ValueBlock:

Тип значения	Описание
Integer	ID типа обратного вызова. В одном варианте осуществления заданы два типа обратных вызовов: ID: RESET = 0 — Все запросы на обратные вызовы, ожидающие рассмотрения, и активные обязательства отменяются после вызова процедуры обратного вызова. Процедура обратного вызова возвращает ESB, которая указывает, может ли, и как, приложение может продолжать текущую операцию. ID: CONTINUE =1 — Вызывается процедура обратного вызова, тогда как все остальные запросы на обратные вызовы, ожидающие рассмотрения, и активные обязательства остаются в силе. Процедура обратного вызова возвращает простой код результата. Приложение может продолжать текущую операцию, если код результата не указывает неудачу.
String	Точка входа в вызов в кодовый модуль. В одном варианте осуществления, это должен быть один из элементов таблицы экспорта кодового модуля для того же объекта управления, который содержит процедуру, возвратившую ESB с этим параметром.
Integer	Куки. Это значение передается в стек вызываемой процедуры.

9.4.1.3. Флаги параметра.

В одном варианте осуществления используются те же самые флаги параметра, которые определены в предыдущем разделе. В одном варианте осуществления, обратные вызовы и обязательства, необходимые вызывающей сущности для реализации, помечаются как CRITICAL, чтобы приложение хоста не имело возможности игнорировать эти параметры.

9.4.2. События.

В одном варианте осуществления события указаны именем. В зависимости от типа события может существовать набор заданных флагов, которые дополнительно описывают событие. В одном варианте осуществления, если для конкретного события не задано никаких флагов, значение поля flag задается равным 0. Кроме того, некоторые события могут указывать, что некоторая информация должна передаваться процедуре обратного вызова при наступлении события. В одном варианте осуществления если от приложения хоста не требуется никакой специальной информации, приложение хоста должно совершать вызов с пустым ArgumentsBlock (см. описание интерфейса процедуры обратного вызова в разделе 3.3, ниже).

В одном варианте осуществления, если приложение хоста не понимает или не поддерживает имя события в параметре обратного вызова, помеченном CRITICAL, приложение хоста должно рассматривать этот параметр как непонятый CRITICAL параметр (и действие, на которое запрошено разрешение, не должно осуществляться).

В одном варианте осуществления заданы следующие имена событий:

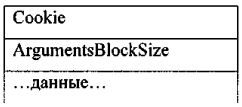
Имя события	Флаги события	Параметр события	Описание
OnPlay	нет	нет	Приложение хоста должно осуществлять обратный вызов, когда начинается воспроизведение мультимедийного объекта.
OnStop	нет	нет	Приложение хоста должно осуществлять обратный вызов, когда воспроизведение мультимедийного объекта останавливается (или приостанавливается)
OnTimecode	нет	Время представления, выраженное числом секунд после начала представления	Приложение хоста должно осуществлять обратный вызов, когда достигнуто или превышено указанное время представления (в ходе нормального воспроизведения в реальном времени или после поиска). Время представления отсчитывается от момента начала представления. Время представления относится к времени медиа-источника, а не к времени настенных часов (например, когда представление приостанавливается, время представления не изменяется).
OnSeek	нет	нет	Приложение хоста должно осуществлять обратный вызов, когда происходит прямой доступ к произвольной точке в представлении мультимедиа. В одном варианте осуществления, при осуществлении обратного вызова, приложение хоста должно обеспечивать следующие данные в ArgumentsBlock: Тип | Кодировка | Описание 32-битовое беззнаковое целое | 4 байта в обратном порядке следования байтов | Смещение позиции поиска 32-битовое беззнаковое целое | 4 байта в обратном порядке следования байтов | Диапазон позиций поиска Позиция в представлении мультимедиа это *смещение* "метки" вне *диапазона* всех "меток" в представлении.

			Например, для представления длительностью 327 секунд, поиск 60-ой секунды можно представить смещением=60, диапазоном=327. Вызывающая сущность выбирает единицу, которая соответствует измерению смещения и диапазона (это может быть единица времени, байт как единица размера или любая другая единица), при условии, что "метки" однородно распределены по всему представлению. Значение *смещения* должно быть меньше или равно значению *диапазона*.

9.4.3. Процедуры обратного вызова.

В одном варианте осуществления процедуры обратного вызова принимают один и тот же вход.

Input: Вершина стека:



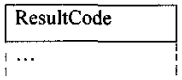
Cookie: значение поля Cookie, указанное в параметре обратного вызова.

ArgumentsBlockSize: число байтов данных, передаваемых в стек под этим параметром. При вызове процедуры, стек содержит значение ArgumentsBlockSize, выданное вызывающей сущностью, указывающее размер блока аргументов на вершине, после которого следует ArgumentsBlockSize байтов данных. В одном варианте осуществления, если размер не кратен 4, данные в стеке дополняются нулевыми байтами, чтобы гарантировать, что указатель стека остается кратным 4.

9.4.3.1. Обратные вызовы CONTINUE.

В одном варианте осуществления обратные вызовы типа CONTINUE (ID типа = 0) имеют следующий выход.

Output: Вершина стека:



ResultCode: целочисленное значение. Значение результата равно 0, если процедура выполнена, или отрицательному коду ошибки, если произошла ошибка.

Описание: если ResultCode указывает, что процедура обратного вызова выполнена (т.е. значение равно 0), приложение хоста может продолжать текущую операцию. Если ResultCode указывает, что произошла ошибка, приложение хоста прерывает текущую операцию и отменяет все ожидающие обратные вызовы и обязательства.

9.4.3.2. Обратные вызовы RESET.

Когда процедура управления указывает один или несколько обратных вызовов типа RESET в ESB, возвращаемой процедурой, приложение хоста вызывает любую указанную процедуру обратного вызова, когда условие для этого обратного вызова выполняется. В одном варианте осуществления, если выполняются условия любого из обратных вызовов, приложение хоста должно:

отменить все остальные ожидающие обратные вызовы;

отменить все текущие обязательства;

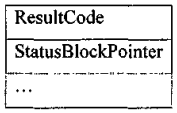
обеспечить все необходимые параметры (если существуют) для этого обратного вызова;

вызвать указанную процедуру обратного вызова.

Состояние возврата из процедуры указывает приложение хоста, если оно может продолжать осуществление текущей операции. В одном варианте осуществления, если в разрешении отказано, или не удается успешно выполнить процедуру, приложение хоста должно прервать осуществление текущей операции. Аналогично, если разрешение дано, приложение хоста должно подчиняться любому обязательству или обратному вызову, которое(ый) может быть возвращен(о) в ESB, как если бы оно вызвало исходную процедуру Control.Actions.<Action>.Perform. Предыдущие обязательства или спецификации обратных вызовов больше не имеют силы.

В одном варианте осуществления все процедуры, указанные как точки входа обратного вызова для этого типа обратного вызова имеют следующий выход.

Output: Вершина стека:



ResultCode: целочисленное значение. Значение результата равно 0, если процедура выполнена, или

отрицательному коду ошибки, если произошла ошибка.

StatusBlockPointer: адрес стандартного ExtendedStatusBlock.

Описание: семантика возврата этой процедуры эквивалентна описанной для процедуры Control.Actions.<Action>.Perform.

9.5. Ресурсы Метаданных.

В одном варианте осуществления объекты управления могут содержать ресурсы метаданных, к которым можно обращаться через параметры, возвращаемые в структурах данных ExtendedStatusBlock. Ресурсы могут представлять собой простой текст, шаблоны текста или другие типы данных. Каждый ресурс идентифицируется посредством ID ресурса и может содержать одну/один или несколько строк текста или элементов кодированных данных, по одной/м для каждой версии на том или ином языке. Не обязательно обеспечивать ресурсы на всех языках. Приложение хоста само решает, версия на каком языке наиболее подходит для его нужд.

Resource		
Поле	Тип	Описание
Id	Строка ASCII	URI (обычно URN, относящийся к Id расширения объекта управления, который содержит кодовый модуль с процедурой, выполняющейся в данный момент)
Type	Строка ASCII	MIME-тип данных ресурса, описанный в IETF RFC 2046
Data	Список LocalizedData	Список всех различных версий ресурса, для разных местных сред

LocalizedData		
Поле	Тип	Описание
Language	Строка ASCII	Код языка, указанный в IETF RFC 3066
Data	Тип зависит от указанного типа mime	Фактические данные для ресурса (текст и т.д.)

Ресурсы сопровождают программы управления, будучи включены в качестве расширений в объект управления. Id ресурса отображается в Id внутреннего расширения объекта управления, который содержит кодовый модуль с процедурой, выполняющейся в данный момент.

С целью вычисления канонической последовательности байтов для объектов Resource, в одном варианте осуществления предусмотрено следующее описание структуры данных:

```
class LocalizedData {
    string language
    byte[] data;
}

class Resource {
string id
string type;
LocalizedData data;
}
```

9.5.1. Простой текст.

Простой текст указан как MIME-тип 'text'.

9.5.2. Шаблоны текста.

Помимо стандартных текстовых ресурсов в одном варианте осуществления задан тип «шаблон текста». MIME-тип для него: 'text/vnd.intertrust.octopus-text-template'.

В одном варианте осуществления шаблон текста содержит символы текста, кодированные в UTF-8, а также именованные заполнители, которые заменяются текстовыми значениями, полученными из параметров, возвращенных в списке параметров, например, для ExtendedStatusBlock. Синтаксис для заполнителя: '\PLACEHOLDER\', где PLACEHOLDER задает имя блока параметра и необязательное указание по форматированию. В одном варианте осуществления, процессор шаблонов должен заменять весь жетон '\PLACEHOLDER\' форматированным представлением поля Value этого блока параметра, и форматирование данных Value указано ниже в разделе 4.2.1.

В одном варианте осуществления, если символ '\' появляется в тексте вне заполнителя, он должен кодироваться как '\\', и процессор шаблонов, встретив в тексте '\\' всегда будет превращать его обратно в '\'.

Синтаксис для заполнителя: FORMAT|NAME, где NAME это имя блока параметра, и FORMAT это указание по форматированию для преобразования данных параметра в текст. Если правил форматирования, принятые по умолчанию, для типа данных значения параметра достаточно, то указание по форматированию можно исключить, и заполнитель будет просто NAME.

9.5.2.1. Форматирование.

9.5.2.1.1. Форматирование по умолчанию.

В одном варианте осуществления, правила форматирования, принятые по умолчанию, для разных типов значения таковы:

Тип	Форматирование
Integer	Текстовое представление целочисленного значения как знакового десятеричного числа. Текст состоит только из символов цифр от “0” до “9” и символа “-“. Если значение равно 0, текст представляет собой строку “0”. Если значение не равно 0, текст не начинается с символа “0”. Если значение отрицательно, текст начинается с символа “-“. Если значение положительно, текст начинается с символа ненулевой цифры.
Real	Текстовое представление значения с плавающей точкой в десятичном исчислении. Целая часть значения представлена с использованием тех же правил, которые используются для целочисленных значений. Десятичный разделитель представлен предпочтительным десятичным разделителем
	приложения хоста. Дробная часть значения состоит из символов “0” в количестве до 6, после которых следует до 3 символов ненулевых цифр.
String	Собственно значение строки
Date	Представление даты, воспринимаемое человеком, согласно предпочтительному для хоста текстовому представлению дат
Parameter	Текст “<name>=<value>”, где <name> это имя параметра, и <value> это значение параметра, форматированное согласно правилам форматирования, принятым по умолчанию для этого типа.
ExtendedParameter	Такое же, как для Parameter
Resource	Текстовая строка данных ресурса. В одном варианте осуществления, ресурс, указанный заполнителем, должен иметь MIMI-тип, который основан на тексте (например, text или text/vnd.intertrust.octopus-text-template).
ValueList	Текст “<value>, <value>, …”, причем все значения в списке форматированы согласно правилам форматирования, принятым по умолчанию для их типа.

9.5.2.1.2. Явное форматирование.

Явные имена форматов можно использовать в качестве части FORMAT тега заполнителя. Если встречается неизвестное имя формата, механизм обработки шаблонов использует правила форматирования, принятые по умолчанию.

Имя	Форматирование
Hex	Шестнадцатеричное представление целочисленного значения, интерпретируемого как беззнаковое. В одном варианте осуществления, это указание по форматированию следует игнорировать для типов данных, которые не являются целыми числами.

9.6. Объекты «контекст».

В одном варианте осуществления, когда процедура управления выполняется, она обращается к ряду объектов «контекст» с использованием системного вызова System.Host.GetObject.

9.6.1. Общий контекст.

В одном варианте осуществления для выполняющихся объектов управления присутствует следующий контекст.

Имя	Тип	Описание
Octopus/Personality/Id	Строка	ID текущего узла индивидуальности
Octopus/Personality/Attributes	Контейнер атрибутов	Атрибуты текущего узла индивидуальности

9.6.2. Контекст среды выполнения.

В одном варианте осуществления следующий контекст присутствует для всех объектов управления, которые выполняются на VM, созданной с использованием системного вызова System.Host.SpawnVm. В одном варианте осуществления, этот контекст не должен существовать или должен быть пустым контейнером для объектов управления, которые выполняются на VM, не созданной с использованием System.Host.SpawnVm.

Имя	Тип	Описание
Octopus/Runtime/Parent/Id	Контейнер безымянных объектов «строка»	Идентичность, под которой выполняется вызывающая сущность системного вызова.

9.6.3. Контекст объекта управления.

В одном варианте осуществления, всякий раз, когда выполняется процедура объекта управления, присутствует следующий контекст:

Имя	Тип	Описание
Octopus/Control/Id	Строка	Id выполняющегося объекта управления
Octopus/Control/Attributes	Контейнер	Атрибуты выполняющегося объекта управления. Этот объект можно опустить, если объект управления не имеет атрибутов.

9.6.4. Контекст объекта «контроллер».

В одном варианте осуществления, всякий раз, когда выполняется процедура объекта управления, и объект управления указан объектом «контроллер» (например, при обращении к объекту ContentKey для потребления защищенного контента), присутствует следующий контекст.

Имя	Тип	Описание
Octopus/Controller/Id	Строка	Id контроллера, который указывает на выполняющийся в данный момент объект управления
Octopus/Controller/Attributes	Контейнер	Атрибуты контроллера, указывающего на выполняющийся в данный момент объект управления. Этот объект можно опустить, если контроллер не имеет атрибутов.

Согласно вариантам осуществления, когда приложению хоста разрешено только группировать ключи контента, которыми управляет единый объект «контроллер», для данного действия, применим лишь один объект «контроллер».

9.6.5. Контекст действия.

В одном варианте осуществления, следующий контекст присутствует всякий раз, когда объект управления вызывается с целью управления действием.

Имя	Тип	Описание
Octopus/Action/Parameters	Контейнер	Массив пар имя/значение, представляющих параметры, относящиеся к текущему действию, если они существуют. В одном варианте осуществления, каждый тип действия определяет список необязательных и обязательных параметров. Этот контейнер можно опустить, если действие не имеет параметров.

9.6.6. Контекст объекта «связь».

В одном варианте осуществления, следующий контекст присутствует всякий раз, когда объект управления вызывается с целью ограничения действительности объекта «связь» (например, объекта управления, внедренного в объект «связь»):

Имя	Тип	Описание
Octopus/Link/Id	Строка	Id объекта «связь»
Octopus/Link/Attributes	Контейнер	Атрибуты объекта «связь», который содержит выполняющийся объект управления. Этот объект можно опустить, если связь не имеет атрибутов.

9.6.7. Контекст агента.

В одном варианте осуществления, следующий контекст присутствует, когда выполняется процедура агента объекта управления:

Имя	Тип	Описание
Octopus/Agent/Parameters	Контейнер	Массив пар имя/значение параметра, представляющих входные параметры агента.
Octopus/Agent/Session/ContextId	Строка	Идентификатор контекста сеанса, в котором выполняется агент.

Контейнеры Parameter и Session обычно используются для того, чтобы протоколы, которые требуют от одной сущности передать агент на другую сущность и запустить его там, могли указать, какие входные параметры передать агенту и какие объекты «контекст сеанса» нужно установить хосту при определенных условиях. Наличие или отсутствие определенных объектов «контекст сеанса» позволяет агентскому коду решить, выполняется ли он как часть протокола, который он призван поддерживать, или же он выполняется вне контекста, в каковом случае он может отказаться выполняться. Например, агент, целью которого является создание объекта «состояние» на хосте, на котором он выполняется, может отказаться выполняться, пока он не будет выполняться в ходе конкретного взаимодействия протоколов.

9.7. Действия.

В одном варианте осуществления, каждое действие имеет имя и список параметров. В одном варианте осуществления, некоторые параметры являются обязательными - приложение должно предоставить их при осуществлении этого действия - и некоторые являются необязательными - приложение может предоставить их, а может и опустить.

В одном варианте осуществления заданы следующие стандартные действия.

9.7.1. Воспроизведение.

Описание. Нормальное воспроизведение мультимедийного контента.

9.7.2. Перенос.

Описание. Перенос на совместимую конечную систему.

Перенос на совместимую конечную систему используется, когда контент нужно сделать доступным системе с той же технологией DRM, чтобы конечная система могла использовать такую же лицензию, как та, которая содержит этот объект управления, но может понадобиться изменить информацию состояния на источнике, приемнике или на обеих системах. Система, с которой осуществляется перенос, называется источником. Конечная система, на которую осуществляется перенос, называется приемником.

Это действие предназначено для использования совместно с протоколом обслуживания, который позволяет переносить агент из источника на приемник для выполнения необходимых обновлений в сохраняемых состояниях источника и приемника (например, объектов в описанной здесь базе данных состояний). В одном варианте осуществления, объект управления использует с этой целью обязательство RunAgentOnPeer. Дополнительная информация об иллюстративных вариантах осуществления этого протокола обслуживания предоставлена ниже в связи с рассмотрением базы данных состояний.

Параметры:

Имя	Тип	Описание
Sink/Id	String	Id узла приемника
Sink/Attributes	Container	Атрибуты узла приемника. Этот контейнер можно опустить, если узел не имеет атрибутов.
TransferMode	String	Transfer Mode ID, указывающий, в каком режиме переносится контент. Этот ID может указывать

<table>
<tr><td></td><td></td><td>стандартный режим, определенный ниже, или URN для собственного режима системы.
В одном варианте осуществления, заданы следующие стандартные режимы:<table><tr><td>ID</td><td>Описание</td></tr><tr><td>Render</td><td>Приемник принимает контент с целью представления</td></tr><tr><td>Copy</td><td>Приемник принимает копию контента</td></tr><tr><td>Move</td><td>Контент переносится на приемник</td></tr><tr><td>CheckOut</td><td>Контент переносится на приемник с блокировкой. Этот режим аналогичен Move с той лишь разницей, что результирующее состояние на приемнике может препятствовать любым другим переносам, кроме переноса обратно на источник.</td></tr></table></td></tr>
<tr><td>TransferCount</td><td>Integer</td><td>Целочисленное значение указывающее, сколько экземпляров счетчиков состояния, связанных с этим объектом управления, нужно перенести на приемник. В одном варианте осуществления, этот параметр является необязательным. Если он отсутствует, переносится только один экземпляр. Он не доложен присутствовать в режимах переноса Render или Copy.</td></tr>
</table>

9.7.3. Экспорт.

Описание. Экспорт в стороннюю конечную систему.

Экспорт в стороннюю конечную систему это действие, которое используется, когда контент нужно экспортировать в system, где нельзя использовать исходную лицензию контента. Это может быть система с другой технологией DRM, система без технологии DRM или система с той же технологией, но в условиях, когда требуется лицензия, отличная от исходной лицензии. Система, из которой осуществляется перенос, называется источником. Конечная система, на которую осуществляется перенос, называется приемником.

В одном варианте осуществления, в результате Extended Status для методов Describe, Check и Perform этого действия, должен быть задан следующий парметр:

Имя	Тип	Описание
ExportInfo	любой	Информация, имеющая значение при экспорте контента в конечную систему, указанную в параметрах действия. Фактические тип и контент этой информации зависят от конкретной конечной системы. Например, для систем на основе CCI, она будет содержать биты CCI, задаваемые для экспортируемого контента.

Параметры:

<table>
<tr><th>Имя</th><th>Тип</th><th>Описание</th></tr>
<tr><td>TargetSystem</td><td>String</td><td>System ID сторонней системы, в которую производится экспорт. Этот ID является URN.</td></tr>
<tr><td>ExportMode</td><td>String</td><td>Export Mode ID, указывающий, в каком режиме экспортируется контент. Этот ID может указывать стандартный режим, определенный ниже, или URN для собственного режима системы.
В одном варианте осуществления заданы следующие стандартные режимы:
<table>
<tr><th>ID</th><th>Описание</th></tr>
<tr><td>DontKnow</td><td>Вызывающая сущность не знает предназначенного режима приемника. В этом случае, программа управления должна предполагать, что приемник может допускать любые разрешенные режимы для TargetSystem, и должна указывать любое ограничение в состоянии возврата процедур действие. Например, для системы на основе CCI, программа управления может возвращать биты CCI, которые разрешают эквивалент Render или Copy в зависимости от того, что разрешает лицензия.</td></tr>
<tr><td>Render</td><td>Приемник принимает контент с целью представления, и не оставляет используемую копию контента, кроме как в целях кэширования, что указывает каждая конечная система</td></tr>
<tr><td>Copy</td><td>Приемник принимает копию контента</td></tr>
<tr><td>Move</td><td>Контент переносится на приемник</td></tr>
</table>
</td></tr>
</table>

Конкретные конечные системы могут требовать других входных параметров.

9.7.3.1. Стандартные конечные системы.

9.7.3.1.1. Аудио-CD или DVD.

В одном варианте осуществления используется стандартный TargetSystem ID 'CleartextPcmAudio', когда конечная система является незашифрованным носителем, на который записан несжатый аудиосигнал ИКМ, например, записываемый аудио-CD или DVD. Для этой конечной системы, параметр ExportInfo является единичным целочисленным параметром, представляющим флаг авторских прав. Этот флаг задается в младшем бите целочисленного значения.

Битовый индекс	Описание
0 (LSB)	При задании этого флага, бит или флаг Copyright должен быть задан в формате записываемого аудиосигнала, если формат поддерживает передачу такого бита или флага.

10. База данных состояний.

Ниже описано защищенное хранилище объектов, которое механизм DRM, согласно предпочтительным вариантам осуществления, может использовать для обеспечения механизма защищенного хранилища состояний. Такое приспособление полезно для обеспечения программ управления, способных считывать и записывать в защищенную базу данных состояний, которая сохраняет от вызова к вызову. Такую базу данных состояний можно использовать для сохранения объектов состояния, например счетчиков

воспроизведения, даты первого использования, совокупного времени представления и/или т.п. В предпочтительном варианте осуществления защищенная база данных реализована в энергонезависимой памяти, например, флэш-памяти на портативном устройстве, или зашифрованной области жесткого диска на ПК. Однако очевидно, что защищенную базу данных можно реализовать на любом подходящем носителе.

Термин "объект", используемый в этом разделе, в целом относится к объектам данных, содержащимся в защищенном хранилище объектов, но не к объектам (например, объектам управления, контроллерам, связям и т.д.), рассматриваемым здесь в других местах; при необходимости провести различие между этими двумя категориями объектов, термин "DRM объект" будет использоваться в отношении объектов, описанных здесь в другом месте (т.е. объектов управления, контроллеров, протекторов, ключей контента, связей, узлов, и т.п.), тогда как термин "объект состояния" будет использоваться в отношении объектов, хранящихся в базе данных состояний. В нижеследующем рассмотрении, мы будем описывать иллюстративную реализацию базы данных состояний, именуемую "Seashell", которая используется в связи с вариантом осуществления механизма DRM Octopus, описанным здесь в другом месте. Однако очевидно, что варианты осуществления описанных здесь систем и способов можно осуществлять на практике без некоторых или всех признаков этой иллюстративной реализации.

10.1. Объекты базы данных.

Хранилище объектов (например, база данных) содержит объекты данных. В одном варианте осуществления, объекты организованы в логическую иерархию, где объекты «контейнер» являются родителями по отношению к содержащимся в них дочерним объектам. В одном варианте осуществления, существует четыре типа объектов: строка, целое число, массив байтов, и контейнер. С каждым объектом связаны метаданные и тип. В зависимости от типа, объект также может иметь значение.

В одном варианте осуществления программы виртуальной машины могут обращаться к объектам состояния с использованием системных вызовов System.Host.GetObject и System.Host.SetObject, и, как описано более подробно ниже, к метаданным объекта можно обращаться с использованием виртуальных имен. В одном варианте осуществления клиенты базы данных могут изменять некоторые поля метаданных (т.е. они являются доступными для чтения/записи (RW)), тогда как другие поля метаданных доступны только для чтения (RO).

В одном варианте осуществления заданы поля метаданных, представленные нижеследующей таблице:

Поле	Тип	Доступность	Описание
Name	Строка	RO	Имя объекта. В одном варианте осуществления в качестве имен объектов разрешены только следующие символы (все остальные зарезервированы): a-z, A-Z, 0-9, '_', '-', '+', ':', '.', '$', '!', '*', ' '
Owner	Строка	RW	Id владельца этого объекта
CreationDate	Беззнаковое 32-битовое целое	RO	Время создания объекта, выраженное числом минут, прошедших с 00:00:00 местного времени 1 января 1970 г.

ModificationDate	Беззнаковое 32-битовое целое	RO	Время последнего изменения объекта, выраженное числом минут, прошедших с 00:00:00 местного времени 1 января 1970 г. Для объектов «контейнер», это время, когда дочерний объект был последний раз добавлен или удален из контейнера. Для других объектов, это время, когда в последний раз было изменено их значение.
ExpirationDate	Беззнаковое 32-битовое целое	RW	Время истечения срока действия объекта, выраженное числом минут, прошедших с 00:00:00 местного времени 1 января 1970 г. Значение 0 указывает, что объект не имеет срока действия.
Flags	32-битовое битовое поле	RW	Набор логических флагов, указывающих свойства объекта.

В одном варианте осуществления, задается флаг метаданных, представленный в нижеследующей таблице:

Битовый индекс	**Имя**	**Описание**
0 (LSB)	PUBLIC_READ	Если задан, указывает, что управление доступом для этого объекта таково, что любой клиент может читать объект и его метаданные.

Как указано выше, в одном варианте осуществления существует четыре типа объектов состояния: строки, целые числа, массивы байтов и контейнеры. В этом варианте осуществления, значение объекта «строка» является строкой символов в кодировке UTF-8, значение объекта «целое число» является 32-битовым целочисленным значением, и значение объекта «массив байтов» является массивом байтов. В

этом варианте осуществления объект «контейнер» содержит ноль или более объектов. Объект «контейнер» является родительским по отношению к объектам, которые он содержит. Содержащиеся объекты являются дочерними объектами контейнера. Все объекты «контейнер», образующие цепь из родителя объекта, родителя и т.д., называются предками объекта. Если объект имеет другой объект своим родителем, этот объект называется потомком объекта-предка.

10.2. Время жизни объекта.

В одном варианте осуществления, время жизни объектов в базе данных состояний подчиняется ряду правил. Объекты можно уничтожать явно или неявно. Объекты также можно уничтожать в порядке очистки базы данных от мусора. Независимо от того, как уничтожается объект, в одном варианте осуществления применяются следующие правила:

ModificationDate для родительского контейнера этого объекта задается равной текущему местному времени.

Если объект является контейнером, все его дети уничтожаются при уничтожении объекта.

10.2.1. Явное уничтожение объека.

Явное уничтожение объекта происходит, когда клиент базы данных запрашивает удаление объекта (см. «Доступ к объекту» на предмет подробностей относительно того, как это можно сделать с использованием системного вызова Host.SetObject).

10.2.2. Неявное уничтожение объекта.

Неявное уничтожение объекта происходит, когда объект уничтожается в результате уничтожения одного из его объектов-предков.

10.2.3. Очистка от мусора.

В одном варианте осуществления, база данных состояний уничтожает любой объект, срок действия которого истек. Считается, что срок действия объекта истек, когда местное время системы, в которой реализована база данных, превышает значение поля ExpirationDate метаданных объекта. Реализация может периодически сканировать базу данных на предмет объектов, срок действия которых истек, и уничтожать их, или может ждать обращения к объекту, чтобы проверить дату окончания его срока действия. В одном варианте осуществления, реализация не должна возвращать клиенту объект, срок действия которого истек. В одном варианте осуществления, при уничтожении объекта «контейнер» (например, вследствие истечения его срока действия), его дочерние объекты также уничтожаются (и все их потомки, рекурсивно), даже если они еще действительны.

10.3. Доступ к объекту.

В одном варианте осуществления, программы виртуальной машины могут обращаться к объектам в базе данных состояний посредством пары системных вызовов: System.Host.GetObject для считывания значения объекта, и System.Host.SetObject для создания, уничтожения, или задания значения объекта.

В одном варианте осуществления, чтобы выглядеть как дерево объектов хоста, база данных состояний "монтируется" под определенным именем в дереве объектов хоста. Таким образом, база данных выглядит как поддерево в более общем дереве объектов хоста. С этой целью в одном варианте осуществления, база данных состояний содержит встроенный корневой объект «контейнер» высшего уровня, который существует всегда. Этот корневой контейнер, по существу, является именем базы данных.

Все остальные объекты в базе данных являются потомками корневого контейнера. Множественные базы данных состояний можно смонтировать в разных местах дерева объектов хоста (для двух баз данных, монтируемых под одним и тем же контейнером хоста, они должны иметь разные имена своего корневого контейнера). Например, если база данных состояний, корневой контейнер которой имеет имя Database1, содержит единственный дочерний объект «целое число» по имени Child1, базу данных можно смонтировать под объектом «контейнер» хоста "/SeaShell", в каковом случае объект Child1 будет виден как "/SeaShell/Database1/Child1". В одном варианте осуществления, доступ к объектам в базе данных состояний регламентирует политика доступа.

10.3.1. Чтение объектов.

Значение объекта можно читать с использованием системного вызова System.Host.GetObject. В одном варианте осуществления базы данных состояний, четыре типа объекта (целое число, строка, массив байтов и контейнер), которые могут существовать в базе данных, отображаются непосредственно на свои эквиваленты в виртуальной машине. К значениям объектов можно обращаться обычным путем, и можно реализовать стандартные виртуальные имена.

10.3.2. Создание объектов.

Объекты можно создавать, вызывая System.Host.SetObject для имени объекта, который еще не существует. Создание объекта осуществляется согласно спецификации системного вызова. В одном варианте осуществления при создании объекта база данных состояний делает следующее:

Задает поле "owner" метаданных объекта равным значению поля "owner" метаданных родительского объекта «контейнер».

Задает поле CreationDate метаданных равным текущему местному времени.

Задает поле ModificationDate метаданных равным текущему местному времени.

Задает поле ExpirationDate метаданных равным 0 (не имеет срока действия).

Задает поле Flags метаданных равным 0.

Задает поле ModificationDate родительского контейнера равным текущему местному времени.

При создании объекта по пути более глубокому, чем существующая иерархия контейнеров, в одном варианте осуществления, база данных состояний неявно создает объекты «контейнер», которые должны существовать для создания пути к создаваемому объекту. В одном варианте осуществления, неявное создание объектов «контейнер» подчиняется тем же правилам, что и явное создание. Например, если существует контейнер "A", не имеющий детей, запрос на задание "A/B/C/SomeObject" приведет к неявному созданию контейнеров "A/B" и "A/B/C" до создания "A/B/C/SomeObject".

10.3.3. Запись объектов.

Значение объекта можно изменять путем вызова System.Host.SetObject для объекта, который уже существует. Если указанный ObjectType не совпадает с ID типа существующего объекта, возвращается ERROR_INVALID_PARAMETER. В одном варианте осуществления, если ID типа равен OBJECT_TYPE_CONTAINER, не нужно указывать никакого значения (ObjectAddress должен быть ненулевым, но его значение будет игнорироваться). При задании существующего объекта, база данных состояний задает ModificationDate объекта равным текущему местному времени.

10.3.4. Уничтожение объектов.

Объекты можно явно уничтожать путем вызова System.Host.SetObject для объекта, который уже существует, со значеним ObjectAddress равным 0. При уничтожении объекта, база данных состояний предпочтительно

задает ModificationDate родительского контейнера равным текущему местному времени;

уничтожает все его дочерние объекты, если уничтожаемый объект является контейнером.

10.3.5. Метаданные объекта

В одном варианте осуществления, к метаданным для объектов базы данных состояний обращаются с использованием системных вызовов System.Host.GetObject и System.Host.SetObject с виртуальными именами. В нижеследующей таблице приведены стандартные и расширенные виртуальные имена, которые могут иметь объекты в одном варианте осуществления базы данных состояний, и указано, как они могут отображаться в поля метаданных.

Виртуальное имя	Тип	Описание
@Name	Строка	Поле Name метаданных объекта
@Owner	Строка	Поле Owner метаданных объекта
@CreationDate	32-битовое беззнаковое целое	Поле CreationDate метаданных объекта
@ModificationDate	32-битовое беззнаковое целое	Поле ModificationDate метаданных объекта
@ExpirationDate	32-битовое беззнаковое целое	Поле ExpirationDate метаданных объекта
@Flags	32-битовое битовое поле	Поле Flags метаданных объекта

В одном варианте осуществления, реализация должна отклонять запрос на установление поля Flags метаданных, если один или несколько неопределенных флагов заданы равными 1. В этом случае возвращаемое значение для System.Host.SetObject равно ERROR_INVALID_PARAMETER. В одном варианте осуществления при чтении поля Flags метаданных, клиент должен игнорировать любой флаг, который заранее не задан, и при задании поля Flags объекта, клиент должен сначала прочитать его существующее значение и сохранить значение любого флага, который заранее не задан (например, в спецификации системы).

10.4. Управление правами собственности и доступом к объектам.

В одном варианте осуществления, всякий раз, когда делается запрос на чтение, запись, создание или уничтожение объекта, реализация базы данных состояний сначала проверяет, имеет ли вызывающая сущность разрешение на осуществление запроса. Политика, которая регламентирует доступ к объектам, основана на концепциях идентичностей и делегирования принципалов. Для реализации политики, доверенная модель, согласно которой действует реализация, должна поддерживать систему обозначений ау-

тентифицированных программ управления. Для этого виртуальная машина обычно имеет кодовый модуль, который содержит программу, снабжаемую цифровой подписью (прямо или косвенно через защищенную ссылку) с помощью секретного ключа пары ключей PKI, и имеет сертификат имени, который связывает имя принципала с ключом подписания; однако, очевидно, что возможны разные способы определения идентичностей программ управления, любой возможный из которых можно использовать.

В одном варианте осуществления, политика доступа к объектам в базе данных состояний состоит из нескольких простых правил:

доступ для чтения значения объекта предоставляется, если идентичность вызывающей сущности совпадает с владельцем объекта, или если флаг PUBLIC_READ задан в поле метаданных Flags объекта;

доступ для чтения значения объекта предоставляется, если вызывающая сущность имеет доступ для чтения родительского контейнера объекта;

доступ для записи значения объекта предоставляется, если вызывающая сущность имеет доступ для записи в родительский контейнер объекта;

доступ для создания или уничтожения объекта предоставляется, если вызывающая сущность имеет доступ для чтения родительского контейнера объекта;

доступ для чтения или записи метаданных объекта (с использованием виртуальных имен) подчиняется той же политике, что и доступ для чтения и записи значения объекта, с дополнительным ограничением, состоящим в том, что в поля, предназначенные только для чтения, нельзя производить запись.

В одном варианте осуществления, когда политика доступа отклоняет запрос клиента, возвращаемое значение системного вызова для запроса равно ERROR_PERMISSION_DENIED.

Корневой контейнер базы данных состояний, предпочтительно, фиксируется при создании базы данных. При создании объекта, значение его поля Owner метаданных задается равным тому же значению, которое имеет поле Owner метаданных его родительского контейнера. Право собственности объекта может изменяться. Для изменения права собственности объекта, значение поля Owner метаданных можно задать путем вызова системного вызова System.Host.SetObject для виртуального имени '@Owner' этого объекта, при условии, что это разрешено правилами управления доступом.

Согласно вариантам осуществления, где программа управления не имеет доступа к объектам, которые не находятся в собственности того же принципала, от имени которого она выполняется, программа управления должна делегировать "сторонним" объекта доступ к программам, загружаемым из кодовых модулей, которые имеют способность выполняться от имени владельца "стороннего" объекта. Для этого, программа управления может использовать системные вызовы System.Host.SpawnVm, System.Host.CallVm и System.Host.ReleaseVm в виртуальной машине управления.

10.5. Протокол передачи лицензии.

Хранение информации состояния в базе данных, например, вышеописанной, позволяет передавать права между устройствами или экспортировать их из домена (например, путем переноса информации состояния на другое устройство). Нижеследующий раздел описывает варианты осуществления протоколов, посредством которых состояние базы данных можно переносить от источника к приемнику. Заметим, что, хотя этот процесс будет относиться к протоколу передачи лицензии, переносится именно состояние базы данных состояний, а не только фактическая лицензия (например, объект управления и т.д.). Протокол называется протоколом передачи лицензии потому, что, в одном варианте осуществления, перенос инициируется выполнением действия «перенос» в программе управления, и потому, что перенос информации состояния позволяет приемнику успешно применять соответствующую лицензию к фрагменту контента.

На фиг. 32 показан пример передачи 3200 лицензии, состоящей из трех сообщений 3202, 3204, 3206. В примере показанный на фиг. 32 приемник 3210 инициирует протокол, направляя запрос 3202 источнику 3212. В одном варианте осуществления, запрос 3202 содержит ID фрагмента контента, подлежащего переносу. Источник 3212 передает ответ 3204 на приемник 3210, содержащий (i) агент, который будет задавать состояние в базе данных состояний приемника 3210, а также (ii) объект(ы) ContentKey, нацеленный(е) на приемник 3210. Согласно фиг. 32, приемник 3210 передает на источник 3212 подтверждение 3206, что агент запущен. Получив Ключ(и) контента и/или фрагмент контента, приемник может затем использовать контент (например, воспроизводить его через громкоговорители, отображать его на экране и/или представлять его каким-то иным образом) в соответствии со связанными с ним объектами управления.

Хотя в некоторых вариантах осуществления можно использовать подход, показанный на фиг. 32, некоторые потенциальные проблемы включают в себя следующее.

Нет возможности превентивно сообщить источнику, что представление закончилось. В одном варианте осуществления, протокол, показанный на фиг. 32, поддерживает два режима, в которых возникают проблемы: (i) Render (представление не останавливается), и (ii) Checkout (нет регистрации). Ввиду этой проблемы, сущности, выполняющие объект управления, могут придти к выдаче таймаутов на переносимых состояниях. Однако это может приводить к неудобствам для потребителя, когда, например, пользователь намеревается представить контент на одном устройстве, но решает, что на самом деле он хочет представить этот контент на другом устройстве: при существующей конструкции, ему, скорее всего,

придется ждать, пока весь фрагмент контента не будет представлен на первом устройстве, прежде чем он может представить его на другом устройстве. Это может быть нежелательным, если контент сравнительно долог (например, является фильмом).

Может быть трудно анализировать лицензию, связанную с Content ID в запросе. В одном варианте осуществления, запрос содержит только Content ID, и источник извлекает лицензию, связанную с Content ID из его базой данных лицензий. Однако этот процесс может быть подвержен ошибкам, поскольку лицензии могут храниться на сменных носителях, и во время применения протокола, конкретная лицензия может отсутствовать, если носитель был удален. Кроме того, даже при наличии лицензии, может быть неудобно осуществлять поиск лицензий в хранилище лицензий. Кроме того, поскольку с набором Content ID может быть связано несколько лицензий, может быть трудно определить, является ли проанализированная лицензия именно той, которая была предусмотрена в запросе.

Невозможно превентивно запросить у программы управления проверку близости. В одном варианте осуществления, набор системных вызовов/обратных вызовов/обязательств не поддерживает способ, которым программа управления может запрашивать проверку близости равноправного устройства. Вместо этого, программа управления можно только считывать значение объекта хоста Octopus/Action/Parameters/Sink/Proximity/LastProbe, в который приложение в ходе переноса загружает значение, которое оно получило из предыдущего выполнения протокола проверки близости. Это может составлять проблему в случае, когда может быть желательно избегать проверки близости, если такая проверка близости не требуется (например, если известно, что приемник находится в определенном домене).

Протокол имеет только три цикла передачи сообщений. Согласно варианту осуществления показанному на фиг. 32, протокол ограничивается тремя циклами передачи сообщений. Это может быть серьезным ограничением, поскольку протокол будет не способен работать в случае, когда обратный вызов OnAgentCompletion возвращает расширенный блок состояний с другим обязательством RunAgentOnPeer. Кроме того, по окончании протокола, приемник не будет знать наверняка, успешно ли выполнен протокол. Кроме того, проверка близости понадобится до передачи ответа (см. предыдущую проблему), но это не нужно в случае, когда источник и приемник находится в одном и том же домене. Кроме того, в протоколе, показанном на фиг. 32, источник передает ключ контента приемнику, не зная, будет ли когда-либо использоваться этот ключ контента.

Нет возможности указать в ESB, необходима ли передача лицензии. Согласно варианту осуществления показанному на фиг. 32, когда клиент DRM оценивает лицензию (например, Control.Actions.Play.Check), непросто указать сущности, записывающей объект управления, что передача лицензии необходима для получения состояния, которое позволит успешно оценить объект управления.

Источник не может инициировать перенос. В протоколе, показанном на фиг. 32, передача лицензии инициируется приемником. Желательно, чтобы источник также мог инициировать перенос.

Усовершенствованные варианты осуществления

Варианты осуществления, описанные ниже, могут разрешать или смягчать некоторые или все вышеописанные проблемы.

Решение проблемы освобождения. В одном варианте осуществления предусмотрена новая операция освобождения. Когда эта операция указана в запросе, Transfer Mode ID задается равным Release. Чтобы клиент осуществлял корреляцию между операциями Render/CheckOut и Release, в запрос добавляется необязательный элемент SessionId (см. нижеследующий раздел). В одном варианте осуществления, при наличии этого элемента, это отражается в дереве объектов хоста для контекста действия «перенос» под SessionId.

Приемник знает, что он отправил этот SessionId в запросе на освобождение, если расширенный блок состояний, который он получил в сообщении Teardown (см. ниже) содержит параметр:

Имя параметра: SessionId;

Тип параметра: String;

Флаг этого параметра задан равным CRITICAL.

Решение проблемы анализа лицензий (повторного анализа запроса). В одном варианте осуществления решение состоит в том, что устройство-приемник помещает пучок(и) лицензий в запрос, что, по существу, гарантирует, что приемник и источник будут выполнять одну и ту же лицензию. Согласно варианту осуществления, показанному на фиг. 32, схема XML для запроса такова:

```
<xs:complexType name="LicenseTransferRequestPayloadType">
 <xs:sequence>
  <xs:element ref="ContentIdList"/>
  <xs:element ref="Operation"/>
  <xs:element ref="oct:Bundle"/>
 </xs:sequence>
</xs:complexType>
```

Когда ContentIdList содержит список Content ID (по одному на каждый трек/поток), идентифицирующих контент, Operation содержит тип операции передачи лицензии и Bundle содержит узел Personality запрашивающей сущности и связанную с ним подпись.

Для ухода от вышеописанной проблемы анализа лицензий, пучок(и) лицензий можно включить в

запрос, например, изменив схему следующим образом:

```
<! -- новые элементы -->
<xs:element name="LicensePart" type="LicensePartType"/>
<xs:complexType name="LicensePartType">
 <xs:sequence>
  <xs:element ref="oct:Bundle" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute name="contentId" use="optional"/>
</xs:complexType>
<xs:element name="License" type="LicenseType"/>
<xs:complexType name="LicenseType">
 <xs:sequence>
  <xs:element ref="LicensePart" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>

<!-- измененный LicenseTransferRequestPayloadType -->
<xs:complexType name="LicenseTransferRequestPayloadType">
 <xs:sequence>
  <xs:element ref="License"/> <!-- определение см. выше -->
  <xs:element ref="Operation"/>
  <xs:element ref="oct:Bundle"/>
  <xs:element name="SessionId" type="xs:string" minOccurs="0"/>
  <xs:element name="NeedsContentKeys" type="xs:boolean" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
```

В этой схеме, элемент ContentIdList заменен элементом License. Этот элемент несет набор элементов LicensePart. Элемент LicensePart несет элемент oct:Bundle, содержащий объекты «лицензия», а также необязательный атрибут ContentId указывающий, что объекты «лицензия» применяются к этому конкретному ContentId. Элемент LicensePart без атрибута ContentId означает, что объекты, содержащиеся в нижележащем пучке, применяются ко всем Content ID (обычно контроллерам и объектам управления).

В одном варианте осуществления необязательный элемент SessionId не может присутствовать, за исключением случая, когда операция представляет собой urn:marlin:core:1-2:service:license-transfer:release в каковом случае он может присутствовать, если параметр SessionId принят в Расширенном блоке состояний соответствующего действия render или checkout (см. выше).

В одном варианте осуществления, необязательный элемент NeedsContentKeys должен присутствовать со значением «ложь», если приемник знает, что он уже способен дешифровать ключи контента. Отсутствие этого элемента означает, что источник должен перешифровать Ключи контента приемника в случае успеха протокола.

В одном варианте осуществления, при получении такого запроса, элемент лицензии обрабатывается следующим образом:

(1) собирают все атрибуты ContentId, найденные в элементах LicensePart;

(2) обрабатывают все элементы Bundle, найденные в элементах LicensePart;

(3) открывают набор ContentId, собранных ранее;

(4) проверяют соответствующие подписи соответствующих объектов;

(5) в необязательном порядке, вызывают метод Control.Actions.Transfer.Check на обработанном объекте управления;

(6) вызывают Control.Actions.Transfer.Perform на объекте управления процесса.

Разрешение программам управления превентивно запрашивать проверку близости приемника. Чтобы позволить программам управления делать это, можно задать новую пару Obligations/Callbacks. В частности, объект управления может поместить обязательство "ProximityCheckSink" в свой расширенный блок состояний. Это указывает приложению, что близость к приемнику должна проверяться. Когда проверка близости произведена, приложение производить обратный вызов объекта управления с использованием обратного вызова "OnSinkProximityChecked".

В одном варианте осуществления, задается обязательство ProximityCheck, которое применимо только в контексте передачи лицензии. В этом варианте осуществления, должно существовать нуль или одно такое обязательство на расширенный блок состояний, и, если оно присутствует, также должен присутствовать обратный вызов OnSinkProximityChecked.

<table>
<tr><th>Имя</th><th>Тип</th><th>Описание</th></tr>
<tr><td>ProximityCheck</td><td>ValueList</td><td>Приложение хоста должно осуществлять протокол проверки близости к устройству-приемнику.
<table>
<tr><th>Тип</th><th>Описание</th></tr>
<tr><td>String</td><td>Id узла индивидуальности, который должен быть проверен на близость</td></tr>
</table>
</td></tr>
</table>

Обратный вызов OnSinkProximityChecked

Имя	Тип	Описание
OnProximityChecke d	ValueList	Приложение хоста должно осуществлять обратный вызов по завершении проверки близости в одном из параметров обязательства.

Тип	Описание
Callback	Процедура для обратного вызова и соответствующий куки.

Разрешение множественных циклов передачи сообщений в протоколе. На фиг. 33 показано изменение протокола, которое допускает множественные циклы передачи сообщений. Согласно варианту осуществления, показанному на фиг. 33, сообщение Setup 3302 может, например, быть таким же, как усовершенствованное сообщение запроса на передачу лицензии, описанное выше в связи с проблемой/решением анализа лицензий.

Согласно фиг. 33, после Setup 3302, приложение запускает объект управления, как объяснено выше, и получает Расширенный блок состояний (ESB). Этот ESB может содержать обязательство RunAgentOnPeer/обратный вызов OnAgentCompletion. В одном варианте осуществления, обязательство RunAgentOnPeer содержит все параметры, которые приложение источника 3312 должно встроить в сообщение RunAgent 3304. Заметим, что в одном варианте осуществления, сообщение RunAgent 3304 также отправляется, если приложение сталкивается с другой парой обратный вызов/обязательство RunAgentOnPeer/OnAgent Completion в Расширенном блоке состояний обратного вызова OnAgentCompletion (после одного или нескольких обменов сообщениями RunAgent/AgentResult).

В одном варианте осуществления, если ESB не содержит обязательство RunAgentOnPeer/обратный вызов OnAgentCompletion, это значит, что нужно отправить сообщение Teardown (см. ниже). Заметим, что этот ESB может содержать обязательство ProximityCheck/обратный вызов OnSinkProximityChecked, в каковом случае осуществляется протокол проверки близости, и результат считывается из ESB проверенного обратного вызова OnSinkProximity до отправки сообщения Teardown.

В одном варианте осуществления, полезная нагрузка сообщение RunAgent 3304 идентично сообщению Response, предусмотренного в предыдущей конструкции, за исключением того, что оно не несет ContentKeyList.

Согласно фиг. 33, после того, как приемник 3310 запустит агент, переданный источником в сообщении RunAgent 3304, приемник 3310 передает сообщение AgentResult 3306 на источник 3312. В одном варианте осуществления, полезная нагрузка сообщения такая же, как в сообщении Confirmation, описанном в связи с фиг. 32.

Согласно фиг. 33, сообщение Teardown 3308 передается приложением-источником 3312, когда расширенный блок состояний обратного вызова OnAgentCompletion не несет никакой пары обратный вызов/обязательство RunAgentOnPeer/OnAgentCompletion, и это значит, что протокол завершен. В одном варианте осуществления, сообщение Teardown 3308 несет два фрагмента информации: (i) описание результата протокола, благодаря чему приемник 3310 знает, успешно ли завершился протокол, указание причины неудачи (дополнительные подробности см. ниже), и (ii) в случае успеха протокола, обновленные объекты ContentKey (ContentKeyList ответа в предыдущем сообщении), если элемент NeedsContentKey сообщения «установка» задан равным «истина» или отсутствует.

В одном варианте осуществления, описание результата протокола фактически представляет собой Расширенный блок состояний (ESB) последнего вызова объекта управления, не несущего никаких пар обязательство/обратный вызов, связанных с агентом.

В случае неудачи, параметры ESB могут указывать на ресурсы. В одном варианте осуществления, эти ресурсы размещены в расширении ResourceList объекта управления, которое отправлено в сообщении Setup.

В случае успеха, в одном варианте осуществления, время жизни кэша указывает, в течение какого времени можно использовать Ключи контента без нового запроса объекта управления.

Пример такого XML-представления ESB показан ниже и может быть добавлен в схему виртуальной машины:

```
<xs:element name="CacheDuration" type="CacheDurationType"/>

<!-- CacheDurationType -->
<xs:complexType name="CacheDurationType">
        <xs:attribute name="type" type="xs:int"/>
        <xs:attribute name="value" type="xs:int"/>
</xs:complexType>

<xs:element name="ExtendedStatusBlock" type="ExtendedStatusBlockType"/>

<!-- ExtendedStatusBlockType -->
<xs:complexType name="ExtendedStatusBlockType">
 <xs:sequence>
    <xs:element ref="CacheDuration"/>
  <xs:element name="Parameters" type="ValueListBlockType" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute name="globalFlags" type="xs:int" default="0" use="optional"/>
 <xs:attribute name="category" type="xs:int" use="required"/>
 <xs:attribute name="subcategory" type="xs:int" use="optional"/>
 <xs:attribute name="localFlags" type="xs:int" use="required"/>
</xs:complexType>
```

Ниже приведен пример случая использования представления в соответствии с вариантом осуществления вышеописанных усовершенствованных механизмов передачи лицензии. В этом примере функция импорта вещания импортирует фрагмент контента со следующей лицензией:

Play: OK, при наличии локального состояния;

Transfer:

Render OK, если приемник находится в домене X, или если приемник находится поблизости. Единовременно можно представлять только один параллельный поток.

Пусть Core DRMClient1 запрашивает разрешение на представление потока контента. Setup Request передается от приемника (Core DRMClient1) на источник (функция BC Import), содержащая следующие параметры:

License: лицензия, связанная с контентом, который приемник хочет представить;

Operation = urn:marlin:core:1-0:service:license-transfer:render

Bundle = Узел индивидуальности приемника.

Получив запрос, приложение-источник наполняет соответствующие объекты хоста и вызывает метод Control.Actions.Transfer.Perform. Ниже показан иллюстративный псевдокод метода, регламентирующего перенос представления:

```
/* псевдокод метода, регламентирующего
   перенос представления */
ESB* TransferRenderPerform(HostObjectTree* t) {
 // проверить блокировку
 if (t->GetObject("SeaShell/.../lock") != NULL) {
  return new ESB(ACTION_DENIED);
 } else {
  // блокировка, ограниченная по времени,
    // которую мы снимем в случае неудачи
  t->SetObject("SeaShell/.../lock", 1);
  t->SetObject("SeaShell/.../lock@ExpirationTime,
            Time.GetCurrent() + 180);
  // возвратить ESB, который содержит обязательство RunAgentOnPeer
  // и обратный вызов OnAgentCompleted
  return new ESB(ACTION_GRANTED,
            new Obligation(RUN_AGENT_ON_PEER,
                  CheckDomainAgent),
            new Callback(ON_AGENT_COMPLETED,
                  RenderAgentCompleted));
 }
}
```

Предположим, представление не заблокировано, и выполняется обязательство RunAgentOnPeer. Сообщение RunAgent передается с объектом управления, содержащим метод CheckDomainAgent. Получив это сообщение, приемник наполняет соответствующие объекты хоста и вызывает метод CheckDomainAgent. Ниже показан иллюстративный псевдокод для CheckDomainAgent:

```
/* псевдокод для CheckDomainAgent */
AgentResult* CheckDomainAgent(HostObjectTree* t) {
 // проверить, доступен ли узел домена
 if (IsNodeReachable("urn:marlin:…:domain2042x")) {
  return new AgentResult(SUCCESS);
 } else {
  return new AgentResult(FAILURE);
 }
```

Предположим, в целях этой иллюстрации, что приемник действительно находится в домене. Затем приемник передает сообщение AgentResult, содержащее результат этого агента. Получив AgentResult, источник вызывает метод обратного вызова. Ниже показан иллюстративный псевдокод для RenderAgentCompleted:

```
/* псевдокод для RenderAgentCompleted */
ESB* RenderAgentCompleted(HostObjectTree* t,
                AgentResult* ar)
{
 if (ar->IsSuccess()) {
  // дать ESB без обязательства/обратного вызова
  // и время жизни кэша
  return new ESB(ACTION_GRANTED, new CacheDuration(0));
 } else {
  // попытаться произвести проверку близости
  return new ESB(ACTION_GRANTED,
          new Obligation(CHECK_PROXIMITY,
          t->GetObject("…/Sink/Id"),
          new Callback(ON_SINK_PROXIMITY_CHECKED,
                ProximityCheckCompleted));
 }
}
```

Мы предположили, что агент успешно проверил принадлежность приемника домену. Сообщение Teardown передается с (i) перешифрованными ключами контента для приемника (с использованием ключей, снабженными узлом «приемник» в запросе Setup), и (ii) ESB, несущим время жизни кэша, указанный выше (0 в этом случае указывает, что приемнику нужно повторно сделать запрос в следующий раз, когда он хочет обратиться к контенту). Когда приемник принимает это сообщение, он знает, что разрешено представлять контент, и имеет необходимые ключи контента.

Теперь предположим, что пользователь хочет представлять контент на другом своем устройстве, DRMClient2. Проблема в том, что контент заблокирован на 180 мин на источнике. К счастью, когда пользователь нажимает STOP на DRMClient1, DRMClient1 инициирует новый протокол передачи лицензии с операцией: Release. Получив запрос, приложение-источник наполняет соответствующие объекты хоста и вызывает метод Control.Actions.Transfer.Perform. Ниже показан иллюстративный псевдокод метода, регламентирующего освобождение переноса:

```
/* псевдокод метода, регламентирующего
  освобождение переноса */
ESB* TransferReleasePerform(HostObjectTree* t) {
 // проверить блокировку
 if (t->GetObject("SeaShell/…/lock") != NULL) {
  t->SetObject("SeaShell/…/lock, NULL); // удалить
  return new ESB(ACTION_GRANTED);
 } else {
  return new ESB(ACTION_DENIED);
 }
}
```

Поскольку в ESB не найдено никакого обязательства/обратного вызова, это означает, что сообщение Teardown отправляется обратно с этим ESB.

Таким образом, этот случай использования представления иллюстрирует, что, в определенных вариантах осуществления, нет необходимости запрашивать DRMClient операции представления на локальную повторную оценку объекта управления, состояние не нужно переносить от источника к приемнику, объект управления может превентивно запрашивать проверку близости, и контент можно освобождать, когда представляющая сущность закончила работать с ним.

11. Сертификаты.

В одном варианте осуществления сертификаты используются для проверки мандата, связанного с криптографическими ключами, до принятия решений на основании цифровой подписи, созданной с помощью этих ключей.

В некоторых вариантах осуществления механизм DRM построен так, чтобы быть совместимым со стандартными технологиями сертификатов, и может пользоваться информацией, найденной в элементах таких сертификатов, например, периодах действия, именах и т.п. Помимо этих основных ограничений, в некоторых вариантах осуществления можно задать дополнительные ограничения в отношении того, для чего можно использовать сертифицированный ключ, а для чего его нельзя использовать. Это можно делать, например, с использованием расширений использования ключа, доступных как часть стандартного

правила кодирования сертификатов. Информация, закодированная в таких расширениях, позволяет механизму DRM проверять, авторизован ли ключ, которым подписан конкретный объект, для использования с этой целью. Например, определенный ключ может иметь сертификат, который позволяет ему подписывать объекты «связь» только, если связь идет от узла с конкретным атрибутом к узлу с другим конкретным атрибутом, но не другую связь. Поскольку семантика общей технологии, используемая для выражения сертификата, обычно не способна выражать такое ограничение, поскольку она не располагает средствами для выражения условий, которые относятся к элементам, например, связям и узлам, характерным для механизма DRM, в одном варианте осуществления такие ограничения для конкретного механизма DRM переносятся как расширение пользования ключом для базового сертификата, которое будет обрабатываться приложениями, которые сконфигурированы для использования механизма DRM.

В одном варианте осуществления ограничения в расширении пользования ключом выражаются категорией пользования и программой ограничения VM. Категория пользования указывает, объекты каких типов ключ авторизован подписывать. Программа ограничения может выражать динамические условия на основании контекста. В одном варианте осуществления любая проверяющая сущность, получающая запрос на проверку действительности такого сертификата, должна понимать семантику механизма DRM, и делегирует оценивание выражения расширения пользования ключом механизму DRM, который использует экземпляр виртуальной машины для выполнения программы. Сертификат считается действительным, если результат выполнения этой программы успешен.

В одном варианте осуществления роль программы ограничения состоит в возвращении логического значения. "Истина" означает, что условия ограничения выполнены, и "ложь" означает, что они не выполнены. В одном варианте осуществления программа управления обращается к некоторой информации контекста, которую можно использовать для принятия решения, например, информации, доступной программе через интерфейс «объект хоста» виртуальной машины. Информация, доступная в качестве контекста, зависит от того, какого типа решение пытается принять механизм DRM, когда он запрашивает проверку сертификата. Например, прежде чем использовать информацию в объекте «связь», в одном варианте осуществления, механизму DRM нужно проверить, что сертификат ключа, которым подписан объект, позволяет использовать этот ключ с этой целью. При выполнении программы ограничения, среда виртуальной машины наполняется информацией, относящейся к атрибутам связи, а также атрибутам узлов, на которые ссылается связь.

В одном варианте осуществления программа ограничения, внедренная в расширение пользования ключом, кодируется как кодовый модуль виртуальной машины, который экспортирует по меньшей мере одну точку входа по имени "Octopus.Certificate.<Category>.Check", где имя "Category" указывает сертификаты какой категории нужно проверить. Параметры для программы проверки проталкиваются в стек до вызова точки входа. Количество и типы параметров, передаваемых в стек, обычно зависит от категории оцениваемого расширения сертификата.

12. Цифровые подписи.

В предпочтительных вариантах осуществления некоторые или все объекты, используемые механизмом DRM, подписываются. Ниже приведено описание, как объекты снабжаются цифровыми подписями в одном варианте осуществления с использованием спецификации цифровой подписи XML (http://www.w3.org/TR/xmldsig-core) ("XMLDSig"). Кроме того, описан метод каноникализации XML, совместимый с эксклюзивной каноникализацией XML (http://www.w3.org/TR/xml-exc-c14n/) ("c14n-ex"), выход которого может обрабатываться анализатором, не знающим пространства имен XML. В Приложении D представлена дополнительная информация по иллюстративной сериализации объектов, включающую в себя иллюстративный способ вычисления канонической последовательности байтов для объектов, не зависящий от кодировки.

Как показано на фиг. 28, 34 и 35, в предпочтительных вариантах осуществления, определенные элементы в лицензии DRM подписываются. Методы, например, показанные на фиг. 28, 34 и 35 полезны для предотвращения или затруднения подделки путем замены компонентов лицензии. Согласно фиг. 34, в предпочтительном варианте осуществления, объект «контроллер» 3402 включает в себя криптографические дайджесты или хэши (или другие подходящие связки) 3405, 3407 объекта ContentKey 3404 и объекта управления 3406, соответственно. Контроллер 3402 сам подписывается с помощью MAC (или, предпочтительно, HMAC, который использует ключ контента) и подписи открытым ключом (обычно поставщика контента или лицензии) 3412. В предпочтительном варианте осуществления, подпись открытым ключом контроллера 3412 сама подписывается с помощью HMAC 3410 с использованием ключа контента. Очевидно, что в других вариантах осуществления, можно использовать другие схемы подписи, в зависимости от нужного уровня безопасности и/или других системных требований. Например, можно использовать разные схемы подписи для подписывания контроллера и/или объекта управления, например PKI, стандартных MAC и/или т.п. Согласно другому примеру можно вычислять отдельную подпись MAC для объекта управления и контроллера, вместо того, чтобы включать дайджест объекта управления в контроллер и вычислять единую подпись MAC контроллера. В порядке еще одного примера контроллер можно подписывать как подписью MAC, так и подписью открытым ключом. Альтернативно или дополнительно, можно использовать ключи, отличные от вышеописанных, для генерации различных под-

писей. Таким образом, хотя на фиг. 28, 34 и 35 показано несколько преимущественных методов подписания в соответствии с несколькими вариантами осуществления, очевидно, что эти методы являются иллюстративными и неограничительными. На фиг. 35 показан вариант осуществления, в котором контроллер ссылается на множественные ключи контента. Согласно фиг. 35, в одном варианте осуществления, каждый из ключей контента используется для генерации HMAC контроллера и подписи PKI.

В одном варианте осуществления режим данных, обработка, входные параметры и выходные для каноникализации XML такие же, как для эксклюзивного канонического XML (c14n-ex) за исключением того, что префиксы пространств имен удалены (пространства имен обозначаются с использованием принятого по умолчанию механизма пространства имен) и внешние сущности не поддерживаются, за исключением символьных сущностей. Первое ограничение предусматривает, что атрибут и его элемент должны находиться в одном и том же пространстве имен.

На фиг. 42 показано соотношение между c14n-ex и иллюстративной каноникализацией XML в одном варианте осуществления, где <xml> является одной действительной XML, и где <xml>' = <xml>" только, если <xml> не имеет внешних сущностей и префиксов пространств имен.

Ниже приведен простой пример упрощенной схемы подписи: однако, в предпочтительном варианте осуществления, используется стандартная каноникализация XML.

original	<n1:elem2 id="foo" xmlns:n0="foo:bar" xmlns:n1="http://example.net" xmlns:n3="ftp://example.org"> <n3:stuff/> </n1:elem2>
processed	<elem2 xmlns="http://example.net" id="foo"> <stuff xmlns="ftp://example.org"/> </elem2>

Элементы подписи, рассмотренные в этом разделе, принадлежат пространству имен XMLDSig (xmlns=http://www.w3.org/2000/09/xmldsig#) и заданы в схеме XML, заданной в спецификации XMLDSig. В одном варианте осуществления, элемент «контейнер» XML-представления объектов DRM представляет собой элемент <Bundle>.

В одном варианте осуществления нужно подписывать следующие объекты:

узлы;

связи;

контроллеры;

Объекты управления (необязательные);

расширения (в зависимости от переносимых ими данных).

В одном варианте осуществления подписи нужно отсоединять и элемент <Signature> должен присутствовать в объекте <Bundle>, который содержит XML-представление объектов, которые нужно подписывать.

В одном варианте осуществления блок <Signature> содержит

элемент <SignedInfo>

элемент <SignatureValue>

элемент <KeyInfo>

В одном варианте осуществления <SignedInfo> встроен в следующие элементы:

<CanonicalizationMethod> В одном варианте осуществления, элемент <CanonicalizationMethod> пуст, и его атрибут Algorithm имеет следующее значение: http://www.w3.org/2001/10/xml-exc-c14n#

<SignatureMethod>

В одном варианте осуществления элемент <SignatureMethod> пуст, и его атрибут Algorithm имеет следующие значения:

http://www.w3.org/2000/09/xmldsig#hmac-sha1 (подпись HMAC)

http://www.w3.org/2000/09/xmldsig#rsa-sha1 (подпись открытым ключом) <Reference>

В одном варианте осуществления может существовать один или несколько элементов <Reference> в блоке <SignedInfo>, если более одного объекта нужно подписывать одним и тем же ключом (например, это может иметь место для объекта Control и Controller).

В одном варианте осуществления при подписывании объекта, значение атрибута 'URI' элемента <Reference> равно ID объекта, на который ссылаются. При подписывании локального элемента XML (например, в случае множественных подписей для метода открытой подписи для объектов Controller), значение URI равно значению атрибута 'Id' элемента, на который ссылаются.

В одном варианте осуществления, когда ссылка указывает на объект, который снабжен дайджестом в ссылке, является не XML-представлением объекта, но его канонической последовательностью байтов. Это преобразование объекта указано в XMLDSig посредством блока <Tranforms>. Поэтому, в одном варианте осуществления, элемент <Reference> внедряет этот блок:

```
<Tranforms>
  <Transform Algorithm="http://www.intertrust.com/octopus/cbs-1_0"/>
</Tranforms>
```

В Приложении D представлена дополнительная информация. В одном варианте осуществления никакие другие блоки <Tranform> не разрешены в качестве ссылок на объект.

В одном варианте осуществления элемент <DigestMethod> пуст, и его атрибут Algorithm имеет следующее значение: http://www.w3.org/2000/09/xmldsig#sha1

Элемент <DigestValue> содержит значение дайджеста в кодировке base64.

<SignatureValue> В одном варианте осуществления, значение подпись является значением в кодировке base64 подписи элемента <SignedInfo>, каноникализированного (ex-c14n) ключом, описанным в элементе <KeyInfo>.

<KeyInfo>

Случай HMAC-SHA1 для подписей объектов Controller

В одном варианте осуществления в этом случае <KeyInfo> имеет только один дочерний элемент: <KeyName>, который указывает ID ключа, использованного для подписи HMAC.

Пример:

```
<KeyInfo>
<KeyName>urn:x-octopus:secret-key:1001</KeyName>
</KeyInfo>
```

Случай RSA-SHA1.

В одном варианте осуществления, в этом случае, открытый ключ, используемый для проверки подписи, переносится в сертификате X.509 v3 и может сопровождаться другими сертификатами, которые могут быть необходимы для завершения пути от сертификата к корню CA.

Эти сертификаты переносятся, в кодировке base64, в элементах <X509Certificate>. Эти элементы <X509Certificate> внедрены в элемент <X509Data>, дочерний по отношению к элементу <KeyInfo>, и появляется последовательно, начиная с сертификата ключа подписи. Сертификат корня обычно опускают.

Пример (для краткости, воспроизводятся не все значения иллюстративных сертификатов; опущенный материал указан многоточиями):

```
<KeyInfo>
 <X509Data>
  <!-- сертификат открытого ключа подписи -->
  <X509Certificate>MIICh...</X509Certificate>
  <!-- промежуточный сертификат доверенного корня -->
  <X509Certificate>MIICo...</X509Certificate>
</X509Data>
</KeyInfo>
```

В одном варианте осуществления, объекты «контроллер» должны иметь по меньшей мере одну подпись HMAC для каждого ContentKey, указанного в их списке управляемых целей. Ключ, используемый для каждой из этих подписей, представляет собой значение ключа контента, содержащегося в указанном объекте ContentKey.

Контроллеры также могут иметь подпись RSA. В одном варианте осуществления, если такая подпись присутствует, эта подпись также появляется как <Reference> в каждой из подписей HMAC объекта. Для этого, в одном варианте осуществления, элемент <Signature> для подписи RSA должен иметь атрибут 'Id', уникальный в охватывающем документе XML, который используется в качестве атрибута 'URI' в одном из элементов <Reference> каждой из подписей HMAC. В одном варианте осуществления, проверяющая сущность должна отвергать подписи RSA, не удостоверенные подписью HMAC.

Пример:

```
<Signature Id="Signature.0" xmlns="http://www.w3.org/2000/09/xmldsig#">
 <SignedInfo>
  <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
  <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <Reference                                                    URI="urn:x-
octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">
   <Transforms>
    <Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0"/>
   </Transforms>
   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>
  </Reference>
 </SignedInfo>

<SignatureValue>mjoyW+w2S9iZDG/ha4eWYD1RmhQuqRuuSN977NODpzwUD02FdsAICVjAcw7f4n
FWuvtawW/clFzYP/pjFebESCvurHUsEaR1/LYLDkpWWxh/LlEp4r3yR9kUs0AU5a4BDxDxQE7nUdqU
9YMpnjAZEGpuxdPeZJM1vyKqNDpTk94=</SignatureValue>
 <KeyInfo>
  <X509Data><X509Certificate>MIICh...</X509Certificate></X509Data> </KeyInfo>
</Signature>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
```

```
<SignedInfo>
 <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
 <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
 <Reference URI="#Signature.0">
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
  <DigestValue>AqPV0nvNj/vc51IcMyKJngGNKtM=</DigestValue>
 </Reference>
 <Reference URI="urn:x-octopus.intertrust.com:controller:1357">
  <Transforms>
   <Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0"/>
  </Transforms>
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
  <DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>
 </Reference>
</SignedInfo>
<SignatureValue>TcKBsZZy+Yp3doOkZ62LTfY+ntQ=</SignatureValue>
<KeyInfo>
 <KeyName>urn:x-octopus.intertrust.com:secret-key:2001</KeyName>
</KeyInfo>
</Signature>
```

13. Протокол проверки близости.

В некоторых вариантах осуществления, может быть желательно ограничивать доступ к контенту, услугам и/или другим системным ресурсам на основании физической близости запрашивающей сущности (например, для применения правил, указывающих, что защищенный фрагмент контента нельзя копировать за пределы домашней сети пользователя, офисного комплекса, и/или т.п.). Ниже описаны варианты осуществления протокола проверки близости, которые обеспечивают безопасность без ненужного препятствования осуществлению самой проверки близости. Протокол проверки близости приспосабливается к приложению в разнообразных контекстах, один из которых, как указано выше, является контекстом объектов управления цифровыми правами; однако очевидно, что описанные ниже системы и способы проверки близости не ограничиваются применением к контексту управления цифровыми правами. Например, без ограничения, представленные здесь методы проверки близости также можно использовать в контексте системе согласования сетевых услуг, например описанной в заявке '551 и/или в любом другом подходящем контексте.

В одном варианте осуществления, проверка близости осуществляется путем измерения времени, которое требуется первому вычислительному узлу для приема ответа от второго вычислительного узла на запрос первого вычислительного узла.

Если промежуток времени меньше заранее заданного порога (что обычно указывает, что второй вычислительный узел находится в пределах определенного физического расстояния от первого вычислительного узла), то проверка близости считается успешной.

Очевидно, что вследствие большого разнообразия различных сетевых соединений, по которым могут передаваться запрос и/или ответ, данный промежуток времени может соответствовать диапазону различных расстояний. В некоторых вариантах осуществления, этот разброс просто игнорируется, и проверка близости считается успешной, если время цикла обмена сообщениями запрос/ответ меньше заранее заданного порога (например, 8 миллисекунд или любой другой подходящий промежуток времени), независимо от того, например, используется ли первое сетевое соединение, что может означать, что запрашивающий и отвечающий узлы действительно относительно отстоят друг от друга. В других вариантах осуществления, можно производить определение типа используемого сетевого соединения, и другие требования к циклу обмена сообщениями можно применять ко всем остальным сетевым соединениям.

В предпочтительном варианте осуществления, проверка близости позволяет анкеру (например, клиенту) проверять близость к цели (например, услуге). В одном варианте осуществления, протокол асимметричен, в том, что анкер генерирует секретное порождающее число, которое используется, и только он один использует защищенный таймер. Кроме того, цель не обязана доверять анкеру. Предпочтительные варианты осуществления проверки близости также криптографически эффективны: в одном варианте осуществления используются только две операции открытого ключа.

Генерация множества R из Q пар на основе порождающего числа S.

В одном варианте осуществления, множество R получается из порождающего числа S согласно следующей формуле: $R_i=H^{2Q-i}(S)$. Здесь H(M) это значение дайджеста хэш-функции H над сообщением M, и $H^n(M)=H(H^{n-1}(M))$ для n>=1 и $H^0(M)=M$. Очевидно, что это просто один иллюстративный метод генерации секрета совместного пользования, и что в других вариантах осуществления можно использовать другие методы, не отходя от его принципов.

В одном варианте осуществления алгоритм, используемый для хэш-функции H, представляет собой SHA1 (см., например, FIPS PUB 180-1. Secure Hash Standard. U.S. Department of Commerce/National Institute of Standards and Technology), хотя очевидно, что в других вариантах осуществления можно использовать другие хэш, дайджест сообщения или функции.

В одном варианте осуществления проверка близости осуществляется следующим образом, где "A" это анкер (например, клиент) и "B" это цель (например, услуга):

(a) A генерирует множество R из Q пар случайных чисел $\{R_0, R_1\}$, $\{R_2, R_3\}$... $\{R_{2Q-2}, R_{2Q-1}\}$, как пока-

зано выше.

(b) A передает B: E(PubB, {Q,S}), где E(Y, X) обозначает шифрование X ключом Y, и PubB обозначает открытый ключ для B в паре открытого/секретного ключей.

(c) B дешифрует {Q,S} и предварительно вычисляет R, как показано выше.

(d) B передает A подтверждение приема, указывающее, что он готов продолжать.

(e) A задает счетчик циклов k равным нулю.

(f) A измеряет T_0 = текущее время, (g) A передает B: {k, R_{2*k}}.

(h) Если значение R_{2*k} верно, B передает в ответ R_{2*k+1}.

(i) A измеряет D = новое текущее время - T_0.

(j) Если B передало A верное значение для R_{2*k+1}, и D меньше заранее заданного порога, то проверка близости считается успешной.

Если k+1<Q, A может выполнить новое измерение, увеличив k и перейдя к этапу (f). Если необходимо осуществить более Q измерений, A может начать с этапа (a) с новым множеством R. Например, в некоторых вариантах осуществления проверку близости можно осуществлять повторно (или заранее заданное число раз) пока не будет получен верный ответ в пределах заранее заданного порога (или если верные ответы принимаются в пределах заранее заданного порога сверх заранее заданного процента последовательности вызовов/ответов), поскольку, даже если два вычислительных узла находятся в необходимой близости друг от друга, аномально медленное сетевое соединение, интенсивный трафик, шум и/или т.п. может привести к задержке ответа от B.

На фиг. 36 показан вариант осуществления вышеописанного протокола, в котором анкер (A) определяет, находится ли цель (B) в приемлемой близости от анкера (A). Например, согласно фиг. 36, A может содержать вычислительный узел 3602, который содержит защищенный контент (например, музыку, видео, текст, программное обеспечение и/или т.п.) и/или материал доступа к контенту (например, связь, ключ и/или т.п.), необходимый удаленному вычислительному узлу (B) 3606 для доступа к защищенному контенту, хранящемуся на вычислительном узле B 3606 или доступному ему. Объекты управления, связанные с контентом или материалом доступа к контенту, могут указывать, что его могут совместно использовать только устройства в определенной близости от узла A 3602 (например, для аппроксимации ограничения распространения контента на домашнюю сеть). Альтернативно или дополнительно, такую политику можно применять на системном уровне вычислительного узла A 3602 (который может, например, содержать менеджер доменов домашней сети или сети предприятия). Таким образом, проверка близости не обязана должна быть условием в программе управления, выполняемой виртуальной машиной; вместо этого она просто может быть чем-то, что вычислительный узел A 3602 требует в качестве предмета операционной политики прежде, чем отправить контент или материал доступа к контенту на вычислительный узел B 3606. Для применения таких объектов управления и/или политик, программное и/или аппаратное обеспечение, выполняющееся на вычислительном узле A 3602, может осуществлять вышеописанный протокол проверки близости каждый раз при подаче запроса на распространение защищенного контента или материала доступа к контенту на вычислительный узел B 3606. Альтернативно или дополнительно, проверка близости можно осуществлять с заранее заданными интервалами (например, раз в день) для определения, находится ли узел B 3606 в необходимой близости, и, в случае успешной проверки близости, можно считать, что узел B 3606 находится в необходимой близости в течение заранее заданного периода (например, до осуществления следующей проверки, до истечения заранее заданного времени, и/или т.п.).

Согласно фиг. 36, когда A и B завершают все начальные этапы установки (например, вышеописанные этапы (a)-(e)) 3604, 3608, A и B начинают защищенный, хронируемый обмен вызовами-ответами (например, на вышеописанных этапах (f)-(i)) 3610, который позволяет A определить, находится ли B в приемлемой близости.

Согласно фиг. 36, в одном варианте осуществления A 3602 передает B 3606 запрос установки 3604, содержащий E (PubB, {Q, S}), т.е. число пар Q, а также секретное порождающее число S для пар, зашифрованное открытым ключом шифрования для B (например, ключом, который B используется в контексте согласования услуг). В одном варианте осуществления, {Q, S} представляет собой поток байтов, сцепляющий Q (1 байт) и S (16 байтов) в сетевом порядке следования байтов. В одном варианте осуществления, шифрование осуществляется с использованием шифрования открытым ключом RSA (например, как описано в B. Kaliski, J. Staddon, PKCS #1; RSA Cryptography specifications Version 2.0. IETF RFC2437. October 1998). В предпочтительном варианте осуществления, A предварительно осуществляет доступ к PubB в порядке инспекции, и его сертификат проверяется. Хотя на фиг. 36 показан ответ установки 3608 от B 3606 к A 3602, в других вариантах осуществления, ответ установки 3608 не используется. Как указано выше, получив запрос установки 3604, B 3606, предпочтительно, предварительно вычисляет множество R, что способствует быстрому ответу на последующие вызовы от A 3602.

Согласно фиг. 36, A 3602 передает B запрос вызова 3612, состоящий из [k, R_{2*k}], т.е. индекса k и соответствующего секрета, вычисленного из порождающего числа. В одном варианте осуществления, [k, R_{2*k}] представляет собой поток байтов, сцепляющий k (1 байт) и R_{2*k} (20 байтов) в сетевом порядке следования байтов, в кодировке base64 для переноса. Согласно фиг. 36, в одном варианте осуществления, B

3606 способен передавать ответ вызова 3614 на A 3602, причем ответ вызова 3614 состоит из R_{2*k+1}, т.е. соответствующего секрета из запроса вызова 3612. В одном варианте осуществления, R_{2*k+1} представляет собой поток байтов R_{2*k+1} (20 байтов) в сетевом порядке следования байтов, в кодировке base64 для переноса.

На фиг. 37 показан пример того, как можно использовать вариант осуществления вышеописанного протокола проверки близости для управления доступом к защищенному контенту. Согласно фиг. 37, предположим, что кабельный или спутниковый поставщик контента имеет политику, позволяющую всем устройствам в заранее заданной близости 3708 от персонального видеомагнитофона (PVR) 3702 пользователя осуществлять доступ к контенту через PVR. Таким образом, например, программное обеспечение менеджера доменов, выполняющееся на PVR 3702, может осуществлять проверку близости на устройстве 3704 и 3706, запрашивая доступ к контенту через PVR 3702. В примере, показанном на фиг. 37, устройство 3706 не находится в пределах близости 3708, заданных политикой поставщика услуг, и получает от PVR 3702 отказ в доступе. Напротив, устройство 3704 находится в пределах близости, и получает доступ, например, принимая контент совместно со связью ограниченного периода действия от устройства 3704 на PVR 3702. Альтернативно или дополнительно, связь может содержать программу управления, которая способна самостоятельно инициировать проверку близости к PVR 3702 и отказывать устройству 3704 в дальнейшем доступе к контенту, если device 3704 перемещается за пределы заранее заданной близости 3708 к PVR 3702.

Соображения безопасности

В предпочтительных вариантах осуществления, следует обратить внимание выполнению некоторых или всех из следующих условий:

цикл, содержащий этапы (f)-(i) не повторяется с одним и тем же значением k для любого множества R;

протокол прерывается, если какая-либо сторона принимает неожиданное сообщение, в том числе:

если B принимает неверное значение для R_{2*k} на этапе (g);

если Q не находится в указанном диапазоне на этапе (a);

если к повторяется в цикле;

если k превышает Q.

Протокол может альтернативно или дополнительно прерываться, если A принимает неверное значение R_{2*k+1} на этапе (h). В других вариантах осуществления допустимо определенное количество неверных ответов от B.

Очевидно, что оптимальные значения для Q и заранее заданный временной порог обычно зависят от уникальных условий конкретного применения (например, скорости сети, важности обеспечения относительно тесной близости и т.д.). Поэтому предпочтительно обеспечивать гибкость реализаций в установлении этих значений. В одном варианте осуществления предполагается, что реализации поддерживают минимальное значение Q, равное 64, и значение порога, равное 8 мс (причем, для некоторых современных скоростей сети, 8 мс может соответствовать расстоянию в несколько миль).

Политики безопасности протокола.

В предпочтительном варианте осуществления, для обмена запросами и ответами не требуется никакой дополнительной защиты. Ввиду размера передаваемых сообщений (например, 20 байтов), и их эффективной случайности (при использовании алгоритма хэширования SHA1 или другого метода), для нарушителя криптографически невыполнимо определение верного ответа, даже если нарушитель способен перехватывать запрос.

Очевидно, что вышеописанные варианты осуществления являются иллюстративными, и что можно предложить многочисленные модификации, не отклоняясь от представленных здесь принципов изобретения. Например, хотя выше описано рекурсивно хэшированное секретное порождающее число, для вызова/ответа можно использовать любой подходящий секрет совместного пользования. В одном варианте осуществления, секрет совместного пользования может просто содержать зашифрованное число/сообщение, передаваемое от A к B, и вызов/ответ может просто содержать обмен между A и B фрагментами числа/сообщения (например, A передает B первый символ сообщения, и B передает A второй символ сообщения, и т.д.). Хотя такому методу может недоставать защищенности, характерной для варианта осуществления, описанного в связи с фиг. 36 (поскольку символ в сообщении гораздо легче разгадать, чем 20-байтовый хэш), в некоторых вариантах осуществления такой уровень безопасности может быть достаточным (в особенности, когда, например, изменчивость сетевых задержек, так или иначе, делает механизм проверки близости весьма грубым средством контроля фактической близости), тогда как в других вариантах осуществления безопасность нужно повысить, осуществляя проверку близости несколько раз, где, хотя любую конкретную цифру или бит сравнительно легко угадать, вероятность того, что нарушитель сможет правильно разгадать данную последовательность цифр или битов, быстро уменьшается с увеличением длины последовательности. В этом варианте осуществления проверку близости можно считать успешной, только если B способен обеспечить свыше заранее заданного количества верных ответов подряд (или свыше заранее заданного процента верных ответов).

В целях иллюстрации и объяснения, ниже представлен дополнительный иллюстративный пример

протокола проверки близости. В этом примере, первое устройство, SRC, осуществляет связь со вторым устройством, SNK, по каналу связи (например, компьютерной сети). Мы хотим иметь возможность безопасно определять, близки ли SRC и SNK друг к другу, что измеряется временем, необходимым SNK для ответа на запрос установления связи от SRC. Вызов или пробное сообщение передается от SRC на SNK, и SNK отвечает ответным сообщением. Период времени между отправкой вызова и приемом ответа называется временем цикла обмена или RTT. Во избежание внесения дополнительной задержки в период времени, которое требуется SNK для вычисления и отправки назад ответа на вызов, обычно желательно, чтобы передача вызовов/ответов была как можно более «легкой». В частности, обычно желательно избегать условий, когда SRC или SNK требует осуществления криптографических операций между отправкой вызова и приемом ответа.

Кроме того, чтобы гарантировать, что только SNK способен создавать правильный ответ на вызов от SRC (например, во избежание атаки перехватчика, когда третья сторона перехватывает вызов от SRC и посылает назад ответ под видом SNK), протокол должен выполняться следующим образом:

(1) SRC создает секрет. Этот секрет состоит из одной или нескольких пар случайных или псевдослучайных чисел.

(2) SRC передает секрет на SNK. Эта часть протокола не зависит от времени. SRC и SNK хранят секрет в тайне. Секрет также передается таким образом, чтобы гарантировать, что его знает только SNK. Для этого обычно предусмотрена передача секрета по защищенному аутентифицированному каналу между SRC и SNK (например, SRC может шифровать секретные данные открытым ключом, про который он знает, что только SNK имеет соответствующий секретный ключ). Секретные данные не обязаны быть вышеописанной(ыми) парой(ами) случайных или псевдослучайных чисел. Даже согласно вариантам осуществления, где используются такие пары, секретные данные, передаваемые на этом этапе, должны лишь содержать достаточно информации, чтобы SNK мог вычислить или вывести значения пар чисел. Например, секретные данные могут быть случайным порождающим числом, из которого можно сгенерировать одну или несколько пар псевдослучайных чисел с использованием генератора псевдослучайных чисел на основе порождающего числа.

(3) Когда SRC знает, что SNK готов принять вызов (например, SNK может отправить сообщение READY после приема и обработки секретных данных), SRC создает сообщение вызова. Для создания сообщения вызова. Например, в предпочтительном варианте осуществления, SRC выбирает одну из пар случайных чисел. Если используется более одной пары, данные сообщения вызова содержат информацию, указывающую, какая пара выбрана, а также одно из двух чисел этой пары.

(4) SRC измеряет значение текущего времени T0. Сразу после этого SRC передает сообщение вызова (шифрование или цифровая подпись не требуется) на SNK и ожидает ответа. Альтернативно, SRC может измерить текущее время T0 сразу после отправки сообщения вызова, хотя, предпочтительно, после осуществления каких-либо соответствующих криптографических операций (например, шифрования, подписывания и/или т.п.).

(5) SNK принимает вызов, на основании которого он может идентифицировать одну из пар, принятых им ранее. SNK удостоверяется, что случайное число в вызове является частью пары, и строит ответное сообщение, которое содержит значение другого случайного числа этой пары.

(6) SNK передает ответное сообщение на SRC (шифрование или цифровая подпись не требуется).

(7) SRC принимает ответное сообщение и измеряет значение текущего времени T1. Время цикла обмена RTT равно T1-T0.

(8) SRC удостоверяется, что число, полученное в ответе, равно другому значению в паре, которая была выбрана для вызова. Если числа совпадают, ответ вызова является успешным, и SRC может быть уверен, что SNK отвечает той степени близости, которая определяется временем цикла обмена. Если числа не совпадают, SRC может прервать протокол или, если совместно используется более одной пары, и существует по меньшей мере одна пара, которая не была использована, возвратиться к этапу (3) и использовать другую пару.

Очевидно, что вышеописанные иллюстративные протоколы проверки близости допускают ряд изменений, не отклоняющихся от его принципов. Например, без ограничения, можно использовать другие криптографические алгоритмы, можно использовать другие секреты совместного пользования и/или т.п.

14. Безопасность.

В практических применениях описанных здесь систем и способов, безопасность можно обеспечивать на различных уровнях и с использованием различных методов. Нижеследующее рассмотрение относится, в основном, к конструкции и принципу работы механизма DRM и соответствующего приложения хоста для использования с целью эффективной регуляции потенциально сложных деловых отношений. Когда механизм DRM и приложение хоста действуют надлежащим образом, осуществляется защита контента от неавторизованного доступа или иного использования за счет применения условий связанной с ним лицензии.

Защиту механизма DRM и/или среды, в которой выполняется механизм DRM (например, приложения и оборудование, с которым он взаимодействует) от злонамеренной подделки или модификации можно осуществлять с использованием любой подходящей комбинации методов безопасности. Например,

можно применять такие криптографические механизмы, как шифрование, цифровые подписи, цифровые сертификаты, коды аутентификации сообщения, и т.п., например, описанные здесь в другом месте, для защиты механизма DRM, приложения хоста и/или другого системного программного обеспечения или оборудования от подделки и/или иных атак, которые могут представлять собой структурные и/или тактические меры безопасности, например, умышленное придание непонятности программному обеспечению, самопроверка, разработка заказных модулей, создание «водяных знаков», противодействие средствам отладки и/или иные механизмы. Иллюстративные примеры таких методов можно найти, например, в патенте США № 6,668,325 B1, Obfuscation Methods for Enhancing Software Security, и в совместно присвоенной патентной заявке США № 11/102,306, опубликованной под номером US-2005-0183072-A1; патентной заявке США № 09/629,807; патентной заявке США № 10/172,682, опубликованной под номером US-2003-0023856-A1; патентной заявке США № 11/338,187, опубликованной под номером US-2006-0123249-A1; и в патенте США № 7,124,170 B1, Secured Processing Unit Systems and Methods, которые все, таким образом, включены сюда посредством ссылки в полном объеме. Альтернативно или дополнительно, можно использовать физические методы безопасности (например, использование относительно недоступной памяти, защищенных процессоров, защищенных блоков управления памятью, режимов работы операционной системы с аппаратной защитой и/или т.п.) для дополнительного повышения безопасности. Такие методы безопасности общеизвестны специалистам в данной области техники, и, очевидно, что можно использовать любую подходящую комбинацию из некоторых, ни одного или всех этих методов в зависимости от нужного уровня защиты и/или особенностей конкретного применения. Таким образом, очевидно, что, хотя здесь описаны определенные механизмы безопасности (например, методы вывода ключа, методы цифровой подписи, методы шифрования, и т.п.) в связи с определенными вариантами осуществления, не все варианты осуществления предусматривают использование этих методов.

Еще одна форма безопасности может обеспечиваться институциональной конструкцией и работой системы, а также юридической и социальной регуляцией ее участников. Например, для получения узла индивидуальности, материала, зашифрованного ключом, защищенного контента, и/или т.п., от устройства или сущности может понадобиться подтвердить свое согласие следовать спецификациям и требованиям системы, может понадобиться пройти процесс сертификации, в ходе которого проверяется согласованность сущности с системными требованиями, и/или т.п. Например, устройству или приложению может понадобиться реализовать механизм DRM таким образом, чтобы он был совместимым с другими реализациями в среде, и/или может понадобиться обеспечить определенный тип или уровень защиты от подделки или иных защитных средств. Можно выдавать цифровые сертификаты, свидетельствующие о согласовании устройства или иной сущности с такими требованиями, и эти сертификаты можно проверять до дачи разрешения устройству или сущности участвовать в системе, или как условие предоставления постоянного доступа.

Ниже представлена дополнительная, неограничительная информация по методам безопасности, которые можно использовать согласно изобретению.

Защита системы

В некоторых вариантах осуществления, разработчик системы может, по своему выбору, использовать сочетание методов обновляемости, отказа и/или исправления для управления рисками и ослабления угроз, которые могут возникать вследствие атак на устройства, приложения и службы или их компрометации. Ниже представлены примеры различных технических механизмов, которые можно использовать для ослабления угроз.

Механизмы обновления можно использовать по меньшей мере в двух различных целях. Во-первых, их можно использовать для передачи обновленной информации доверенным системным сущностям, которая позволяет им отказывать в доступе или обслуживании недоверенным системным сущностям. Во-вторых, механизмы обновления позволяют недоверенной сущности восстановить доверенное состояние путем обновления любого скомпрометированного компонента. Отказные контрмеры можно дополнительно охарактеризовать как демонстрирующие один или несколько из следующих режимов поведения:

Отмена или аннулирование мандата (обычно, путем занесения какой-либо сущности в черный список);

Исключение или блокировка доступа путем применения криптографических механизмов или механизмов применения политики;

Бойкот или блокировка доступа или обслуживания на основании идентичности или какого-либо другого атрибута, привязанного к мандату;

Прекращение действия или аннулирование мандата или привилегии на основании временного события.

Например, механизмы отказа можно использовать для противодействия таким угрозам, как клонирование устройства, маскировка под законного пользователя, сбои протокола, сбои в применении политики, сбои в защите приложений и устаревшая или подозрительная информация.

В нижеследующей таблице приведены примеры потенциальных угроз, некоторых рисков, которые они представляют, и механизмов противодействия угрозам и обновления защиты системы.

Угроза	Риски	Механизм исправления	Механизм обновления
Клонирование устройства	Свободный доступ к устройствам.	Шифрование вещания	Обновление BKB.
Скомпрометированный сертифицированный ключ	Неавторизованные лицензии, связи, состояние устройства, идентичности, доступ к службе.	Отмена сертификата	Распространение CRL. Обновление ключа.
Сбой реализации	Средство взлома устройства.	Указание версии спецификации	Обновление программного обеспечения
Сбой протокола	Скомпрометированные ключи. Нерегулируемый доступ к лицензированному контенту.	Задание метаданных безопасности	Обновление программного обеспечения
Устаревшие метаданные безопасности	Фальшивое взаимодействие служб. Обратный ход часов, опора на скомпрометированную информацию.	Задание метаданных безопасности	Служба обновления метаданных безопасности. Обновление программного обеспечения.

Отмена

Отмену можно рассматривать как механизм исправления, опирающийся на занесение сущности в черный список. Обычно отменяется мандат, например, сертификат открытого ключа. После отмены мандата, черный список нужно обновить и использовать механизм обновления для переноса обновления, чтобы заинтересованная сторона могла извлечь из него пользу.

Таким образом, например, можно потребовать, чтобы устройства, пользователи и/или другие сущности представили сертификаты идентичности, другой мандат и различные данные безопасности, прежде чем предать им информацию, необходимую для потребления контента или услуги. Аналогично, чтобы клиент доверял службе, можно потребовать, чтобы служба представила свой мандат клиенту.

Примеры того, как сущность может эффективно сделать недействительной информацию, необходимую для доступа к службе, включают в себя:

списки отмены сертификата (CRL);

службы действительности мандата и данных, например, ответчик протокола Online Certificate Status Protocol (OCSP);

команды самоуничтожения мандата и данных.

Списки отмены сертификата (CRL)

Различные сущности могут использовать списки отмены для отмены сертификатов идентичности, лицензий, связей и других утверждений безопасности. Этот механизм наиболее эффективен для исправления ситуации, которая возникает вследствие компрометации службы. Можно использовать ряд методов для распространения CRL. Например, некоторые системы могут применять косвенный CRL, т.е. существует единственный CRL, управляющей всей экосистемой. Кроме того, сущности могут объявлять (или публиковать) имеющиеся у них CRL и/или подписаться на услугу обновления. CRL можно виртуально распространять между равноправными устройствами, и/или портативные устройства могут получать опубликованные CRL, будучи привязанными. С этой целью также можно использовать методы согласования услуг, описанные в заявке '551.

Службы действительности

Службы действительности можно использовать для обеспечения обновленной информации о состоянии мандата и других данных, связанных с безопасностью. Службы действительности могут осуществлять либо операции активного удостоверения от имени заинтересованной стороны, либо могут использоваться для управления информацией безопасности от имени заинтересованной стороны. Примером активной службы действительности является служба, которая может проверять действительность мандата или атрибута. Примерами служб действительности, которые управляют информацией безопасности, являются службы, которые распространяют CRL или обновления политики безопасности, или обеспечивают защищенную службу времени. Использование служб действительности позволяет гаран-

тировать, что заинтересованные стороны имеют обновленные данные для принятия решений по управлению.

Обычно не всем системным сущностям требуется ежеминутно обновляемая информация о действительности мандата и данных безопасности. Например, не все потребительские устройства используют службу Online Certificate Status Protocol (OCSP) для удостоверения цепи сертификатов сервера лицензий каждый раз при использовании лицензии или получении новой лицензии. Однако сервер лицензий может достаточно часто использовать службу OCSP для проверки действительности мандата подписчика. Политика (которую легко обновлять) может определять, когда и какие службы нужно использовать. Благодаря возможности динамически обновлять политику, серверы лицензий могут адаптироваться к изменениям условий работы. Таким образом, политика безопасности может развиваться на основании опыта, технологического прогресса и рыночных факторов.

Принудительное самоуничтожение объектов безопасности

Самоуничтожение мандата и данных сущностью имеет смысл, когда целостность защитных процессов сущности не вызывает сомнений. При наличии такой возможности, это, зачастую, является наиболее прямым, быстрым и эффективным методом отмены. Это может быть особенно полезно, при наличии малого или, вовсе, отсутствия подозрений в нарушении целостности, и когда двусторонняя связь поддерживает протокол, допускающий конкретные команды уничтожения совместно с проверкой, что это уничтожение было произведено.

Существует ряд объектов безопасности, которые часто бывает полезно уничтожать или отключать. Например, когда устройство покидает домен, или заканчивается срок действия лицензии контента, бывает полезно уничтожить соответствующие объекты, которые содержат ключи и которые можно использовать для доступа к контенту. Агентские программы управления, более подробно описанные в другом месте, весьма пригодны для реализации механизмов самоуничтожения. Агенты можно приспособить для уничтожения состояния в защищенном хранилище (например, базе данных состояний) для актуализации изменений в доменной принадлежности или для удаления ключей, которые уже невозможно использовать (например, вследствие изменений в принадлежности или политике).

Исключение

Исключение это механизм исправления, который препятствует злонамеренному пользователю (или группе злонамеренных пользователей) участвовать в дальнейшем потреблении товаров и услуг. В силу суровых последствий исключения, оно обычно используется как последнее средство, когда этого требуют обстоятельства. Исключение базируется на механизме, который эффективно заносит в черный список злонамеренных пользователей, тем самым запрещая им потреблять медиа-ресурсы и связанные с ними услуги. Распространение черного списка опирается на механизм обновления для обеспечения этого исправления. Однако исключение не обязательно обеспечивает механизм обновления для восстановления злонамеренного пользователя в доверенное состояние.

Исключение ключа

Исключение ключа это механизм управление ключами, который используется для рассылки информации ключа множеству приемников таким образом, чтобы в любое данное время можно было принять решение на логическое лишение некоторого подмножества приемников возможности дешифровать контент в будущем. Этот механизм активируется с использованием эффективных методов построения Блока рассылки ключей (ВКВ), который включает в себя информацию, необходимую каждому члену большой группы приемников для дешифрования контента. ВКВ имеет структуру, позволяющую легко обновлять его, чтобы лишить одного или нескольких членов группы возможности дешифровать контент. Иными словами, конструкция ВКВ позволяет управляющей сущности обновлять систему новым ВКВ, что позволяет поставщику контента избирательно исключать нужное множество устройств из пользования ВКВ, даже если оно может обращаться к нему.

Этот механизм особенно эффективен против атаки клонирования, когда пират обращает инженеров к законному устройству, извлекает его ключи и затем использует копии этих ключей для клонирования устройств. Клоны внешне действуют как оригинал, за исключением того, что эти клоны не всегда следуют модели администрирования. Когда компрометация выявлена, можно распространить обновленный ВКВ, который исключает скомпрометированное устройство и все его клоны. Однако исключение ключа влечет за собой некоторую избыточную нагрузку, связанную с хранением, переносом и вычислением, что, в ряде случаев, делает его менее эффективным по сравнению с другими методами. Это особенно верно, когда контент не подлежит вещанию, или когда существует обратный канал.

Бойкот

Бойкот это механизм исправления, действующий аналогично исключению, но с менее суровыми последствиями. По существу, это средство отказа в обслуживании по решению политики среды выполнения. Вместо более тяжеловесных подходов к лишению устройства дееспособности путем принудительного самоуничтожения или блокировки доступа посредством исключения ключа, бойкот предусматривает простой подход к отключению устройства за счет того, что поставщики услуг отказываются предоставлять ему услуги. При современной тенденции к повышению значимости устройств за счет использования услуг, предоставляемых извне, бойкот становится более эффективным механизмом безопасно-

сти.

Бойкот устройства инициируется политикой, и его можно использовать для обеспечения дискриминационных мер против сущностей (например, клиентов, серверов и конкретных ролевых игроков) которые не предъявляют все необходимые мандаты, которые требует политика. Политика, например, может требовать, чтобы сущность продемонстрировала, что она получила последнее обновление безопасности. Поэтому бойкот может быть следствие либо отмены, либо неудачной попытки совершить какое-либо конкретное действие. Бойкот в среде взаимодействия равноправных устройств с использованием можно обеспечить с помощью инспекционных служб и таких служб, которые, например, описаны в заявке '551. Кроме того, служба сертификации данных (например, экземпляр службы действительности) может осуществлять бойкот в момент применения политики. Когда системная сущность бойкотирована, ей можно сообщить, какой конкретный мандат или объект не отвечает требованиям политики службы. Это может побудить бойкотируемую сущность к обновлению объекта через соответствующий интерфейс службы.

Прекращение действия

Прекращение действия это механизм исправления, который опирается на некоторое временное событие для объявления мандата или объекта недействительным. Прекращение действия эффективно при предоставлении временного доступа к медиа-ресурсам или медиа-услугам; по истечении его срока действия, модель администрирования гарантирует, что доступ более не разрешен. Для эффективного использования прекращения действия могут потребоваться механизмы обновления, позволяющие обновить мандат или объект для обеспечения продолжения доступа к медиа-ресурсам или медиа-услугам.

Прекращение действия мандатов

Сертифицированные ключи имеют различные атрибуты прекращения действия, присвоенные для защиты заинтересованных сторон. Прекращение действия мандат можно использовать для гарантии того, что сущностям, срок действия сертификатов которых истек, будет отказано в обслуживании, и использовать совместно с процедурами пролонгации ключа и обновления ключа. Когда предполагается, что сущности часто подключаются к глобальной сети, с практической точки зрения целесообразно регулярно обновлять мандат и другие данные безопасности. Другая практическая мера состоит в том, чтобы сделать период действия этих объектов как можно короче. В политиках проверки действительности можно использовать различные методы, например, перекрывающиеся периоды действия и льготные периоды, чтобы обеспечить непрерывность работы во время транзакций. Короткие периоды действия также помогают уменьшить размер CRL.

Прекращение действия связей

Как описано выше, объектам «связь» можно присваивать периоды действия. По истечении срока действия, связь считается недействительной, и механизм DRM не учитывает ее при построении своего графа. Этот механизм можно использовать для обеспечения временного доступа к товарам и услугам. Связи можно обновлять, чтобы непрерывный доступ к медиа-ресурсам можно было предоставлять, пока это разрешает политика. Поскольку, в одном варианте осуществления, связи являются сравнительно легкими, самозащищенными объектами, их легко распространять по протоколам взаимодействия равноправных устройств.

Механизмы обновляемости: обновляемость приложений и политик

Эффективная обновляемость обычно обеспечивает быстрое применение исправлений к сбоям протокола, которые часто являются основными проблемами безопасности, возникающими в сфере обеспечения безопасности (в том числе, в системах DRM). Обновления программного обеспечения можно использовать для обновления бизнес-логики и протоколов безопасности. Когда приложения приспособлены к политике безопасности и политике доверия, отдельно от логики приложения, можно использовать отдельный механизм для обновления политики; это менее рискованный подход. Фактически, можно использовать механизмы публикации между равноправными устройствами для быстрого обновления политики. В противном случае, можно использовать методы обновления программного обеспечения разработчика приложения для обновления политик безопасности и доверия.

Использование инструмента прав для правовой работы

Обычно желательно использовать относительно легкие инструменты, когда возможно. Использование мандатов с ограниченными периодами действия и политик, которые проверяют даты окончания срока действия, способствует поддержанию общей совокупности сущностей в управляемых пределах и избавляет от необходимости в слишком быстром увеличении размеров CRL. Бойкот сущности вместо лишения ее доступа к ключам может увеличивать срок действия ВКВ; кроме того, его преимущество состоит в обеспечении дифференцируемых политик, которые могут иметь временный характер и изменяться в зависимости от обстоятельств. Разные CRL, которые отслеживают конкретные типы мандата, представляющие интерес для различных ролевых игроков, можно использовать вместо ВКВ, которые можно распространять там, где они наиболее эффективны (например, имея дело с клонированными приемниками). Политики могут предписывать использование онлайновых служб действительности, когда предполагается, что эти службы обеспечивают разумную компенсацию затрат времени и усилий, когда свежие мандаты очень важны, и когда более медленные механизмы отмены непригодны. Когда узел, предположительно, обладает целостностью и совершает правильные действия, и когда объект лицензии или безо-

пасности (например, связь для подписки или связь домена) подлежит отмене, то разумный подход обычно состоит в предписании узлу уничтожить объект. В такой ситуации, нет необходимости объявлять лицензию недействительной и нет необходимости распространять ВКВ или повторно снабжать домен ключом. Самоуничтожение, инициируемое локальной политикой или самостоятельной командой, является одним из более эффективных методов отмены.

Очевидно, что хотя были описаны разнообразные технологии отмены, обновления, исправления и другие технологии и практики, очевидно, что для разных ситуаций используются разные инструменты, и что предпочтительные варианты осуществления описанных здесь систем и способов можно практически применять с использованием любой подходящей комбинации из некоторых или ни одного из этих методов.

Защита сетевых служб

В нижеследующем рассмотрении представлены некоторые соображения и методы безопасности, которые могут относиться к вариантам осуществления, в которых вышеописанные механизм DRM и приложения используются в связи с системами и способами согласования сетевых служб, например, описанными в заявке '551.

Практические реализации систем DRM, используемые механизмом и архитектурой DRM, например, раскрытыми здесь, часто осуществляют сетевые транзакции для доступа к контенту и объектам DRM. В таком контексте, системы и способы, описанные в заявке '551, можно использовать, в том числе, для стандартизации защиты на уровне сообщений, включающей в себя аутентификацию сущностей и форматы атрибутов авторизации (ролей).

В целях рассмотрения, транзакции, которые происходят в системе DRM, можно разделить на по меньшей мере две общие категории на основании типа информации, к которой осуществляется доступ, которую получают, или над которой производятся манипуляции.

Транзакции доступа к контенту предусматривают прямой доступ или манипулирование мультимедийным или развлекательным контентом или другой важной информацией, защищаемой системой DRM. Примеры транзакций доступа к контенту включают в себя представление защищенного видеоклипа, создание копии защищенного аудиотрека на компакт-диске, перемещение защищенного файла на портативное устройство, передачу конфиденциального документа по электронной почте и т.п. Транзакции доступа к контенту обычно предусматривают прямой доступ к ключу защиты контента и осуществляются на месте потребления по требованию пользователя.

Объектные транзакции это транзакции, в которых пользователь или система получает или взаимодействует с объектами, заданными системой DRM, которая тем или иным образом регламентирует доступ к защищенному контенту. Такие объекты включают в себя лицензии DRM, жетоны принадлежности, списки отмены и т.д. Одна или несколько объектных транзакций обычно необходимы прежде, чем будет доступно все обеспечение, необходимое для осуществления транзакции доступа к контенту. Объектные транзакции обычно характеризуются использованием того или иного типа сети связи для сборки объектов DRM на месте потребления.

Эти два типа транзакций задают два пункта управления, которые, в общем случае, относятся к большинстве систем DRM. На фиг. 38 показана типичная пара взаимодействий, в которых клиент 3800 с наличием DRM запрашивает лицензию DRM 3802 у соответствующей службы 3804 лицензий DRM. В примере, показанном на фиг. 38, лицензия DRM 3802 передается от службы 3804 лицензий DRM на клиент 3800, где она оценивается для обеспечения доступа к контенту 3806.

Системы DRM обычно требуют, чтобы транзакции доступа к контенту и объектные транзакции осуществлялись таким образом, что препятствовать неавторизованному доступу к контенту и создавать объекты, защищающие контент. Однако вопросы безопасности для двух типов транзакций принципиально различны. Например: транзакции доступа к контенту могут требовать аутентификации принципала-человека, проверки защищенного счетчика представления, оценивания лицензии DRM для вывода ключа защиты контента, и т.д. Основной угрозой законному выполнению транзакции доступа к контенту является нарушение границы, устойчивой к подделке, которая защищает объекты и данные внутри.

Объектные транзакции обычно предусматривают канал связи между сущностью, которая требует объект DRM, и сущностью, которая его обеспечивает. По этой причине объектные транзакции сталкиваются с угрозами на основе связи, например атаками перехватчика, атаками ретрансляции, атаками отмены услуг и атаками, в которых неавторизованные сущности получают объекты DRM, которые они не могут законно обрабатывать.

В общем случае, объектные транзакции предусматривают аутентификацию двух взаимодействующих сущностей, защиту сообщений, передаваемых между ними, и авторизацию транзакции. Основной целью таких транзакций является сбор объектов DRM с защищенной целостностью из законных источников, что позволяет осуществлять транзакции доступа к контенту. С точки зрения транзакции доступа к контенту, механизмы, посредством которых получаются законные объекты DRM и дополнительная информация, используемая при их получении, по существу, не имеют значения; эти механизмы могут (и, предпочтительно, должны) быть невидимыми для самого доступа к контенту. Это принципиальное разделение вопросов приводит, в предпочтительном варианте осуществления, к многоуровневой модели

связи, которая отличает доверенную инфраструктуру связи от приложений, надстроенных на ней.

Упрощенный пример получения лицензии и потребления, показанный на фиг. 38, затемняет некоторые детали, которые обычно играют важную роль в практическом применении. Например, в этом примере не показано, как служба лицензий DRM удостоверяется, что сущность, запрашивающая лицензию DRM, действительно является законным клиентом DRM, а не злонамеренной сущностью, пытающейся получить неавторизованную лицензию или заблокировать обслуживание законных клиентов путем потребления полосы сети и мощности обработки. Также не показано, как осуществляется защита важной информации в отношении конфиденциальности и целостности при ее переносе по каналам связи, соединяющим клиент и службу.

Эта иллюстративная транзакция более подробно показана на фиг. 39. На фиг. 39 пунктирная линия представляет логическую транзакцию с точки зрения клиента 3800 представления контента и сервера 3804 лицензий DRM на уровне приложений. Стек 3900, изображенный под ними, представляет уровни обработки, используемые для обеспечения доверенной и защищенной доставки между двумя конечными точками.

Согласно фиг. 39, клиент представления 3800 запрашивает лицензию 3802 у сервера 3804 лицензий DRM. Пунктирная линия на этой схеме указывает, что оригинальный источник и конечный потребитель информации представляют собой клиент 3800 представления контента и сервер 3804 лицензий DRM. Однако, на практике, полезная нагрузка сообщения фактически может обрабатываться на нескольких уровнях обработки, находящихся между логикой уровня приложений и незащищенным каналом связи 3902, соединяющим две конечные точки.

Уровни обработки, которые отделяют компоненты уровня приложений от незащищенного канала связи, совместно называются стеком безопасности. Стек безопасности можно рассматривать как защищенную инфраструктуру обмена сообщениями, которая гарантирует, конфиденциальную передачу, с защитой целостности, сообщений между доверенными конечными точками. Многоуровневая модель стека обеспечивает следующие преимущества:

(1) Конструкторам логики уровня приложений не нужно тратить усилий на разработку нижележащих механизмов защищенной связи, которые соединяют конечные точки. Доверенный инфраструктуры обмена сообщениями является общим шаблоном конструкции, который, будучи однажды построен, может быть распространен на многие разные ситуации независимо от логики уровня приложений, которая их поддерживает.

(2) Сама инфраструктура обмена сообщениями может оставаться независимой от конкретной семантики передаваемых ею сообщений и упора, который она делает на предупреждение атак, относящихся к связи, и атак на аутентичность конечных точек обмена сообщениями.

В одном варианте осуществления, стек безопасности состоит из нескольких разных уровней обработки, которые описаны ниже. В одном варианте осуществления системы и способы согласования услуг, описанные в заявке '551, можно использовать для обеспечения некоторых или всех операций стека безопасности.

Аутентификация

В одном варианте осуществления, конечные точки обмена сообщениями можно аутентифицировать. Аутентификация это процесс, в котором данная конечная точка демонстрирует другой, что ей дано действительное имя органом, которому доверено делать это. Опорная конечная точка в транзакции должна доверять органу присвоения имен; установление такого органа обычно осуществляется организациями, применяющими доверенную технологию.

Общий механизм для демонстрации обладания действительного имени использует криптографию открытого ключа и цифровые подписи. Согласно этому подходу сущность снабжается тремя фрагментами информации:

(1) различимое имя, которое обеспечивает идентификатор сущности;

(2) пара асимметричных ключей, состоящая из открытого ключа и секретного личного ключа; и

(3) сертификат, снабженный цифровой подписью, который утверждает, что держателю секретного ключа дано различимое имя.

Сертификат связывает различимое имя и секретный ключ. Сущности, которая использует секретный ключ для подписывания фрагмента информации, доверено иметь данное различимое имя. Подпись можно проверить с использованием только открытого ключа. Например, аутентификация может производиться на основании стандарта X.509v3.

Поскольку в одном варианте осуществления сущности, которая может продемонстрировать обладание сертифицированным секретным ключом, доверено иметь различимое имя, указанное в сертификате, защита секретного ключа, используемого для подписывания информации, приобретает особую важность. Таким образом, возможность использования личного ключа для подписывания задает границы сущности, идентифицируемой различимым именем. На уровне приложений отправители и получатели должны знать, что сообщения отправлены доверенными сторонами. Поэтому, в одном варианте осуществления важно, чтобы логика уровня приложений сама была частью аутентифицированной сущности. По этой причине в одном варианте осуществления стек безопасности и уровни приложений, которые опираются

на него, предпочтительно заключены в границу доверия, в связи с чем предполагается, что подсистема, содержащаяся в границе доверия, обобществляет доступ к личному ключу подписывания сообщений сущности.

Авторизация

Вышеописанный механизм аутентификации доказывает распределенным конечным точкам обмена сообщениями, что идентичность их корреспондента заслуживает доверия. Во многих вариантах осуществления эта информация является слишком грубой - более подробная информация о возможностях и свойствах конечных точек может потребоваться для принятия политических решений в отношении определенных транзакций. Например, в контексте фиг. 38, клиенту представления контента может понадобиться знать, не только, что он осуществляет связь с аутентифицированной конечной точкой, но также, осуществляет ли он связь со службой, которая считается уполномоченной обеспечивать действительные объекты «лицензия» DRM.

Варианты осуществления стека безопасности обеспечивают механизм для объявления, переноса и применения политики, который основан на более дифференцируемых атрибутах аутентифицированных сущностей, посредством механизма авторизации. С использованием этого механизма, сущностям, которые уже обладают мандатом аутентификации, присваиваются утверждения ролей, которые связывают именованное множество возможностей с различимым именем сущности. Например, имена ролей можно задавать для клиента DRM и сервера лицензий DRM.

Именованные роли призваны переносить конкретные возможности, которыми владеет сущность. На практике, роли можно присоединять к сущности, объявляя связь между различимым именем сущности и именем роли. Как и сертификаты аутентификации, которые связывают ключи с различимыми именами, в одном варианте осуществления, объявления ролей, используемые для авторизации, подписываются доверенным органом объявления роли, который может отличаться от генератора имен. Внутри сущности, объявления ролей проверяются совместно с мандатом аутентификации как условие предоставления доступа к уровню приложений конечной точки обмена сообщениями.

Сущность может носить столько атрибутов роли, сколько необходимо для построения приложения. Пример, показанный на фиг. 40, демонстрирует сущность с множественными ролями: одной ролью, которая указывает способность функционировать в качестве клиента DRM, и двумя служебными ролями. Например, одна сущность может быть одновременно клиентом DRM, поставщиком объектов DRM и поставщиком данных безопасности. В одном варианте осуществления, SAML 1.1 используется для объявлений, касающихся атрибутов сущности.

Защита сообщений

Нижним уровнем стека безопасности является уровень защиты сообщений, который обеспечивает целостность, конфиденциальность и актуальность защиты сообщений, и снижает риск атак на канал связи, например, атак ретрансляции. На уровне защиты сообщений:

сообщения между процессами уровня приложений подписываются с использованием личного ключа подписывания сообщений сущности, что обеспечивает защиту целостности и устойчивость к атакам перехватчика;

сообщения шифруются с использованием открытого ключа, которым владеет сущность-адресат. Это гарантирует, что непредназначенные получатели не могут читать сообщения, перехваченные при переносе;

в сообщение добавляются примечания и метки времени, обеспечивающие иммунитет к атакам ретрансляции и облегчающие испытания на живучесть между конечными точками обмена сообщениями;

Используются метки времени сервера для обновления доверенного времени механизма DRM.

В одном иллюстративном варианте осуществления, предусмотрена поддержка симметричного шифрования AES, криптографии открытого ключа RSA, дайджестов подписи SHA-256, и механизмов для сигнализации других алгоритмов в сообщениях.

15. Протокол автозагрузки.

В некоторых вариантах осуществления протокол автозагрузки используется для доставки начальных конфиденциальных данных конфигурации на сущности, например, устройства и программные клиенты. Например, когда сущность желает войти в более крупную сеть или систему и осуществлять связь с другими сущностями с использованием криптографических протоколов, может потребоваться конфигурирование ее персонализованными данными, включающими в себя набор ключей (совместного пользования, секретного и открытого). Когда для невозможно или непрактично предварительно конфигурировать сущности персонализованными данными, ей приходится "самозагружаться" с использованием криптографического протокола.

Описанный ниже иллюстративный протокол использует секрет совместного пользования в качестве основания для автозагрузки сущности с набором ключей и другими данными конфигурации. В нижеследующих разделах будет использоваться следующая система обозначений:

E(K, D) - шифрование некоторых данных D ключом K;

D(K, D) - дешифрование некоторых зашифрованных данных D ключом K;

S(K, D) - подпись некоторых данных D ключом K. Это может быть подпись открытым ключом или

MAC.

H(D) - дайджест сообщения для данных D.

V(K, D) - проверка подписи на некоторых данных D ключом K. Это может быть проверка подписи открытым ключом или MAC.

CertChain(K) - цепь сертификатов, связанная с открытым ключом K. Значение K включено в первый сертификат цепи.

CertVerify(RootCert, CertChain) - проверка того, что цепь сертификатов CertChain (включающая в себя открытый ключ, найденный в первом сертификате цепи) действительна под корневым сертификатом RootCert.

A | B | C | ... - последовательность байтов, полученная связыванием отдельных последовательностей байтов A, B, C,....

CN(A) - каноническая последовательность байтов для A.

CN (A, B, C,...) - каноническая последовательность байтов для сложных полей A, B, C ....

15.1. Начальное состояние.

15.1.1. Клиент.

В одном варианте осуществления клиент имеет следующий набор жетонов автозагрузки (заранее загруженных во время изготовления и/или в программно-аппаратном/программном обеспечении).

Один или несколько сертификатов, предназначенных только для чтения, которые являются корнем доверия для процесса автозагрузки: BootRootCertificate.

Один или несколько секретных ключей аутентификации автозагрузки: BAK (совместного пользования).

Необязательный секретный ключ генерации порождающего числа автозагрузки (уникальный для каждого клиента) BSGK. Если клиент имеет хороший источник случайных данных, это порождающее число не нужно.

Некоторая информация ClientInformation, которую клиент должен передать службе автозагрузки для получения своего ключа конфиденциальности (например, ClientInformation может включать в себя серийный номер устройства, имя изготовителя и т.д.). Эта информация состоит из списка атрибутов. Каждый атрибут является парой (имя, значение).

Клиент можно конфигурировать множественными сертификатами BootRootCertificate и ключами аутентификации BAK, чтобы он имел возможность участвовать в протоколе загрузки с различными серверами загрузки, которые могут требовать разных доверенных доменов.

15.1.2. Сервер.

В одном варианте осуществления сервер имеет следующие жетоны:

по меньшей мере один из ключей аутентификации автозагрузки клиента: BAK (секрет совместного пользования);

пара открытого/секретного ключей, используемая для подписи: (Es,Ds);

цепь сертификатов ServerCertificateChain = CertChain(Es), которая действительна под одним из корневых сертификатов: BootRootCertificate;

пара открытого/секретного ключей, используемая для шифрования: (Ee/De).

15.2. Описание протокола.

Иллюстративный вариант осуществления протокола автозагрузки показан на фиг. 41 и описан ниже. Неудача при выполнении процесса (например, при проверке подписи или цепи сертификатов) приведет к ошибке и остановке выполнения протокола.

BootstrapRequestMessage.

Клиент передает на сервер запрос, указывающий, что он хочет инициировать сеанс автозагрузки, и обеспечивает некоторые начальные параметры (например, версию протокола, профиль и т.д.), а также ID сеанса (для предотвращения атак ретрансляции) и список доверенных доменов, в которых он может участвовать. В нижеследующей таблице показан иллюстративный формат для BootStrapRequestMessage:

Имя	BootstrapRequestMessage		
Атрибуты	Имя	Описание	
	Protocol	Символическое имя протокола	
	Version	Версия протокола	
	Profile	Имя профиля для этих протокола/версии	
Направление	Клиент → Сервер		
Полезная нагрузка	BootstrapRequest		
	Имя	Тип	Описание
	SessionId	Строка	Уникальный ID сеанса, выбранный клиентом
	TrustDomains	Список строк	Имена всех доверенных доменов, в которых клиент может участвовать
Предполагаемый ответ	ChallengeRequestMessage		

Атрибуты Protocol и Version сообщения указывают, какую спецификацию протокола использует клиент, и поле Profile идентифицирует заранее заданный набор криптографических протоколов и форматов кодирования, используемый для обмена сообщениями и данными.

Клиент выбирает SessionId, который должен быть уникальным для этого клиента и не подлежит повторному использованию. Например, для генерации уникального ID сеанса можно использовать уникальный ID для клиента и значение возрастающего счетчика.

В одном варианте осуществления клиент также передает список всех доверенных доменов, для которых он был сконфигурирован.

В одном варианте осуществления, сервер принимает BootstrapRequestMessage и осуществляет следующие этапы:

Проверяет, поддерживает ли он указанные Protocol, Version и Profile, запрошенные клиентом.

Генерирует Nonce (строго случайное число).

В необязательном порядке генерирует Cookie для переноса информации например, метки времени, жетона сеанса или любой другой информации со стороны сервера, которая будет сохраняться на протяжении сеанса. Значение куки имеет смысл только для сервера, и рассматривается клиентом как непрозрачный блок данных.

Извлекает значение SessionId из BootstrapRequestMessage.

Генерирует вызов: Challenge = [Nonce, Ee, Cookie, SessionId].

Вычисляет S (Ds, Challenge) для подписывания вызова с помощью Ds.

Строит ChallengeRequestMessage и передает его обратно на клиент в порядке ответа.

ChallengeRequestMessage.

В нижеследующей таблице показан иллюстративный формат для ChallengeRequestMessage:

Имя	ChallengeRequestMessage		
Направление	Сервер → Клиент		
Полезная нагрузка	Challenge		
	Имя	Тип	Описание
	Nonce	Последовательность байтов	Случайный нонс, генерируемый сервером
	ServerEncryptionKey	Последовательность байтов	Закодированный открытый ключ Ee, используемый для шифрования полезной нагрузки сообщения
	Cookie	Последовательность байтов	Непрозрачные данные, генерируемые сервером
	SessionId	Строка	ID сеанса, генерируемый клиентом
	Signature	Последовательность байтов	Закодированная цифровая подпись S(Ds, CN(Challenge)) канонической последовательности байтов вызова Canon(Challenge) = CN(CN(Nonce), CN(ServerEncryptionKey), CN(Cookie), CN(SessionId))
	ServerCertificateChain		
	Имя	Тип	Описание
	TrustDomain	Строка	Доверенный домен, в котором цепь сертификатов действительная
	Certificates	Список последовательностей байтов	Список закодированных сертификатов, образующих цепь: CertChain(Es). Первый сертификат в массиве сертифицирует открытый ключ Es, и каждый из последующих сертификатов, в свою очередь, сертифицирует открытый ключ предыдущего сертификата. Последний сертификат в массиве имеет открытый ключ, сертифицированный корневым сертификатом CA для доверенного домена
В ответ на	BootstrapRequestMessage		

В одном варианте осуществления, получив ChallengeRequestMessage, клиент осуществляет следующие этапы.

Удостоверяется, что цепь сертификатов ServerCertificateChain действительная под корневым сертификатом BootRootCertificate: CertVerif(BootRootCertificate, ServerCertificateChain).

Извлекает открытый ключ Es из ServerCertificateChain.

Проверяет подпись вызова: V(Es, Challenge).

Проверяет, совпадает ли SessionId с идентификатором сеанса, выбранным для сеанса при отправке BootRequestMessage.

Строит ChallengeResponseMessage и передает его на сервер.

ChallengeResponseMessage.

Для генерации ChallengeResponseMessage клиент осуществляет следующие этапы.

Генерирует сеансовый ключ SK с использованием одного из следующих методов:

прямого, с использованием защищенного генератора случайных ключей;

косвенного, с использованием Nonce и BSGK: вычисляет HSK = H (BSGK | Nonce), и задает SK = первые N байтов из HSK;

генерирует объект ChallengeRepsonse, который содержит [Challenge, ClientInformation, SessionKey]. Здесь, Challenge входит в состав ранее принятого ChallengeRequestMessage, где ServerEncryptionKey опущен;

вычисляет S (BAK, ChallengeResponse) для подписывания ответа с помощью BAK;

шифрует подписанный ChallengeReponse с помощью SK: E (SK, [ChallengeResponse, S(BAK, ChallengeResponse)]);

шифрует SessionKey открытым ключом сервера Ee.

строит ChallengeResponseMessage и передает его на сервер

<table>
<tr><td>Имя</td><td>ChallengeResponseMessage</td></tr>
<tr><td>Направление</td><td>Клиент → Сервер</td></tr>
<tr><td>Полезная нагрузка</td><td>
SessionKey [зашифрованный с помощью Ee]
<table>
<tr><td>Имя</td><td>Тип</td><td>Описание</td></tr>
<tr><td>SessionKey</td><td>Последовательность байтов</td><td>Закодированный сеансовый ключ SK, зашифрованный открытым ключом сервера Ee</td></tr>
</table>
ChallengeResponse [зашифрованный с помощью SK]
<table>
<tr><td>Имя</td><td>Тип</td><td>Описание</td></tr>
<tr><td>Challenge</td><td>Объект</td><td>
Challenge
<table>
<tr><td>Имя</td><td>Тип</td><td>Описание</td></tr>
<tr><td>Nonce</td><td>Последовательность байтов</td><td>Случайный нонс, генерируемый сервером</td></tr>
<tr><td>Cookie</td><td>Последовательность байтов</td><td>Непрозрачные данные, генерируемые сервером</td></tr>
<tr><td>SessionId</td><td>Строка</td><td>Уникальный ID сеанса</td></tr>
</table>
</td></tr>
<tr><td>ClientInformation</td><td>Массив атрибутов</td><td>
Массив из 0 или более объектов «атрибут»:
<table>
<tr><td>Attribute</td></tr>
</table>
<table>
<tr><td>Имя</td><td>Тип</td><td>Описание</td></tr>
<tr><td>Name</td><td>Строка</td><td>Имя атрибута</td></tr>
<tr><td>Value</td><td>Строка</td><td>Значение атрибута</td></tr>
</table>
</td></tr>
<tr><td>SessionKey</td><td>Последовательность байтов</td><td>Закодированное значение секретного сеансового ключа SK</td></tr>
<tr><td>Signature</td><td>Последовательность байтов</td><td>Закодированная цифровая подпись S(BAK, CN(ChallengeResponse)) канонической последовательности байтов CN(ChallengeResponse) = CN(CN(Challenge), CN(ClientInformation), CN(SessionKey))</td></tr>
</table>
</td></tr>
<tr><td>Предполагаемый ответ</td><td>BootstrapResponseMessage</td></tr>
</table>

Сервер принимает BootstrapChallengeResponse и осуществляет следующие этапы:

дешифрует сеансовый ключ SK с использованием своего секретного ключа De: D(De, SessionKey);

дешифрует ChallengeResponse сеансовым ключом SK, полученным на предыдущем этапе: D(SK, Challenge);

проверяет подпись вызова: V(BAK, ChallengeResponse);

проверяет, совпадает ли сеансовый ключ SK с ключом, используемым для дешифрования;

проверяет, при необходимости, значения Cookie и Nonce (например, метку времени);

проверяет, совпадает ли SessionId с идентификатором сеанса, выбранным для сеанса при отправке BootRequestMessage;

строит BootstrapResponseMessage и предает его на клиент.

BootstrapResponseMessage.

Для генерации BootstrapResponseMessage, сервер осуществляет следующие этапы:

анализирует ClientInformation, полученную в ChallengeResponseMessage и ищет или генерирует данные;

конфигурации клиента, которые нужно передать для этого запроса автозагрузки (они могут включать в себя ключи конфиденциальности (Ec/Dc) для узла, представляющего клиент). Обычно сервер использует значение Nonce и Cookie для облегчения извлечения верной информации для клиента.

создает BootstrapResponse с SessionId и данными конфигурации;

вычисляет S (Ds, BootstrapResponse) для подписывания Data с помощью Ds;

шифрует подписанный BootstrapResponse сеансовым ключом SK: E(SK, [BootstrapResponse, S(Ds, BootstrapResponse)])

Имя	BootstrapResponseMessage		
Направление	Сервер → Клиент		
Полезная нагрузка	BootstrapResponse [зашифрованный с помощью SK]		
	Имя	Тип	Описание
	SessionId	Строка	ID сеанса
	Data	Последовательность байтов	Данные конфигурации для клиента
	Signature	Подпись	Цифровая подпись S(Ds, CN(BootstrapResponse)) канонической последовательности байтов CN(BootstrapResponse) = CN(CN(SessionID), CN(Data))
В ответ на	ChallengeResponseMessage		

15.3. Доверенные домены.

В одном варианте осуществления каждый доверенный домен включает в себя орган корневого сертификата и уникальное имя домена. Когда клиент передает BootstrapRequest, он идентифицирует все доверенные домены, которые он желает принимать (т.е. сертификаты которых он будет считать действительными). Сервер выбирает доверенный домен из списка, переданного клиентом, если он поддерживает хотя бы один из них.

15.4. Подписи.

В одном варианте осуществления всякий раз при использовании подписей в полезной нагрузке сообщений, подписи вычисляются на канонической последовательности байтов для полей данных, содержащихся в подписанной(ых) части(ях) сообщения. Каноническая последовательность байтов вычисляется из значений полей, но не из кодировки значений полей. Каждый профиль, предпочтительно, задает алгоритм, используемый для вычисления канонической последовательности байтов полей для каждого типа сообщения.

15.5. Профили.

Профиль протокола автозагрузки это набор вариантов различных криптографических шифров и форматов сериализации. Каждый профиль, предпочтительно, имеет уникальное имя и включает в себя следующие варианты:

алгоритм шифрования открытым ключом;

алгоритм подписи открытым ключом;

алгоритм шифрования секретным ключом;

алгоритм подписи секретным ключом;

кодирование открытым ключом;

алгоритм дайджеста;

каноническая сериализация объекта;

формат сертификата;

минимальный размер нонса;

маршализация сообщений.

Приложение А

Ниже приведен пример объекта «контроллер» с множественными блокирующими подписями. Примечание: в этом примере ключи контента не шифруются


```
<Controller xmlns="http://www.intertrust.com/Octopus/1.0" id="urn:x-
octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">
<ControlReference>
 <Id>urn:x-octopus.intertrust.com:control:0001</Id>
<Digest>
 <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
 <DigestValue
xmlns="http://www.w3.org/2000/09/xmldsig#">1z95n10V7CBiKs/rSQdXvKyZmfA=</DigestValue>
 </Digest>
 </ControlReference>
<ControlledTargets>
<ContentKeyReference>
 <Id>urn:x-octopus.intertrust.com:контент-key:2001</Id>
 </ContentKeyReference>
<ContentKeyReference>
 <Id>urn:x-octopus.intertrust.com:контент-key:2002</Id>
 </ContentKeyReference>
<ContentKeyReference>
 <Id>urn:x-octopus.intertrust.com:контент-key:2003</Id>
 </ContentKeyReference>
 </ControlledTargets>
 </Controller>
<Signature Id="Signature.0" xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
 <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
 <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
<Reference URI="urn:x-octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">
<Transforms>
 <Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0" />
 </Transforms>
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
 <DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>
 </Reference>
 </SignedInfo>

<SignatureValue>mjoyW+w2S9iZDG/ha4eWYD1RmhQuqRuuSN977NODpzwUD02FdsAICVjAcw7f4nFWuvt
awW/clFzYP/pjFebESCvurHUsEaR1/LYLDkpWWxh/LlEp4r3yR9kUs0AU5a4BDxDxQE7nUdqU9YMpnjAZE
GpuxdPeZJM1vyKqNDpTk94=</SignatureValue>
<KeyInfo>
<X509Data>

<X509Certificate>MIIC6jCCAlOgAwIBAgIBBjANBgkqhkiG9w0BAQUFADCBszELMAkGA1UEBhMCVV
MxEzARBgNVBAgTCkNhbGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMSAwHgYDVQQKExdJb
nRlcnRydXN0IFRlY2hub2xvZ2llczEUMBIGA1UECxMLT2N0b3B1cyBEUk0xGDAWBgNVBAMTD09jdG9
wdXMgVGVzdCBDQTEnMCUGCSqGSIb3DQEJARYYb2N0b3B1cy10ZXN0LWNhQDhwdXMubmV0MB4
XDTA0MDQwODAwNTUyOVoXDTA0MDUwODAwNTUyOVowgcExCzAJBgNVBAYTAlVTMRMwEQY
DVQQIEwpDYWxpZm9ybmlhMRQwEgYDVQQHEwtTYW50YSBDbGFyYTEgMB4GA1UEChMXSW50ZX
J0cnVzdCBUZWNobm9sb2dpZXMxFDASBgNVBAsTC09jdG9wdXMgRFJNMR8wHQYDVQQDExZPY3Rv
cHVzIFRlc3QgTm9kZSAwMDAxMS4wLAYJKoZIhvcNAQkBFh9vY3RvcHVzLXRlc3Qtbm9kZS0wMDAxQ
DhwdXMubmV0MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDU8AJQArJg+VTuwUO2fMv5sCt
fmZECyJjA0vbgQc+cPXpfeIdACiCL1n1eml/ZLlu7ZaRwQeo1yJSeK57bxv+zhW14F1jnqS/IKLG84RG1eoMiO
T1lhErb2nU3xT0KCgxsEXFAbfwAYnLX7hpy/1ho2mTmJbgksWoPrPw3xMPCYwIDAQABMA0GCSqGSIb3
DQEBBQUAA4GBAH1rHStXcQkFmcYhl5zck6twsNIRF+/1HZGuTGKeb6+J2ZLk6sNUWXLOID1oPRMde7
X1RiqpDNkbG4xoPoxHiK9VdfBstjv9Q8iUceziMIXVV/q+XJMd7HfBJq25XqBScS9/RAKKKwuRRkQHEV3u
BABvLSCzIRSJH9bFuYzNeVne</X509Certificate>
```

```
 </X509Data>
 </KeyInfo>
 </Signature>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
 <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
 <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
<Reference URI="#Signature.0">
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
 <DigestValue>AqPV0nvNj/vc51IcMyKJngGNKtM=</DigestValue>
 </Reference>
<Reference URI="urn:x-octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">
<Transforms>
 <Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0" />
 </Transforms>
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
 <DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>
 </Reference>
 </SignedInfo>
 <SignatureValue>TcKBsZZy+Yp3doOkZ62LTfY+ntQ=</SignatureValue>
<KeyInfo>
 <KeyName>urn:x-octopus.intertrust.com:secret-key:2001</KeyName>
 </KeyInfo>
 </Signature>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
 <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
 <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
<Reference URI="#0">
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
 <DigestValue>AqPV0nvNj/vc51IcMyKJngGNKtM=</DigestValue>
 </Reference>
<Reference URI="urn:x-octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">
<Transforms>
 <Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0" />
 </Transforms>
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
 <DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>
 </Reference>
 </SignedInfo>
 <SignatureValue>qAunQpXC18kl8Veo8UHbcXTqHCA=</SignatureValue>
<KeyInfo>
 <KeyName>urn:x-octopus.intertrust.com:secret-key:2002</KeyName>
 </KeyInfo>
 </Signature>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
 <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
 <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
<Reference URI="#0">
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
 <DigestValue>AqPV0nvNj/vc51IcMyKJngGNKtM=</DigestValue>
 </Reference>
<Reference URI="urn:x-octopus.intertrust.com:controller:37A50262EE3389A14ABC0BC7BE5D43E5">
<Transforms>
 <Transform Algorithm="http://www.intertrust.com/Octopus/xmldsig#cbs-1_0" />
 </Transforms>
 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
 <DigestValue>G1zXF9Sz/zCwH6MaFm0ObOQcxuk=</DigestValue>
 </Reference>
 </SignedInfo>
 <SignatureValue>bRxLSM82d4ktWsYz6uhBxzJfsOo=</SignatureValue>
<KeyInfo>
 <KeyName>urn:x-octopus.intertrust.com:secret-key:2003</KeyName>
 </KeyInfo>
 </Signature>
 </Bundle>
```

Приложение B

В этом приложении B представлено XML-кодирование объектов в одном варианте осуществления системы с использованием иллюстративного механизма DRM Octopus, описанного здесь в другом месте. Для конкретного приложения, можно создать схему XML, зависящую от приложения, путем импорта схемы XML, показанной ниже ("Octopus XML Schema") и добавления элементов, зависящих от приложения (например, расширений, используемых для отмены). В одном варианте осуществления, кодировка объектов в XML должна быть способно проходить удостоверение согласно схеме XML, зависящей от приложения. Дополнительные возможные ограничения на эти XML-кодировки представлены ниже.

В примере, представленном в этом приложении B, базовым типом схемы XML для всех объектов DRM является OctopusObjectType. Это означает, что все объекты поддерживают атрибуты и расширения. Тип каждого элемента объекта Octopus является производным этого базового типа. Эти типы могут группировать другие элементы, например, элемент SecretKey для ContentKeyType.

В этом иллюстративном варианте осуществления ключи системы распространения ключей Scuba описаны применительно к исключению: элемент ScubaKeys является дочерним по отношению к элементу расширения. То же самое касается ключей отмены с расширением Torpedo.

Как описано здесь в другом месте, существуют разные виды объектов Octopus (например, ContentKey, Protector, Controller, Control, Node и Link). Эти объекты можно связывать друг с другом совместно с расширениями с использованием элемента <Bundle>. В одном варианте осуществления, если объекты или расширения подписаны в <Bundle>, то <Bundle> содержит элементы <Signature>, описанные здесь в другом месте.

Схема XML Octopus (Octopus.xsd):

```
<?версия xml ="1.0" кодировка="UTF-8"?>
<xs:schema targetNamespace="http://intertrust.com/Octopus/1.0"
xmlns="http://intertrust.com/Octopus/1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<!-- импорты -->
```

```
        <xs:import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="xmldsig-core-schema.xsd"/>
        <xs:import namespace="http://www.w3.org/2001/04/xmlenc#" schemaLocation="xenc-schema.xsd"/>
        <!-- элементы верхнего уровня -->
        <xs:element name="RootLevelObject" type="RootLevelObjectType" abstract="true"/>
        <xs:element name="OctopusObject" type="OctopusObjectType" abstract="true"/>
        <!-- базовый элемент -->
        <xs:element name="Bundle" type="BundleType"/>
        <xs:element name="Link" type="LinkType" substitutionGroup="RootLevelObject"/>
        <xs:element name="Node" type="NodeType" substitutionGroup="RootLevelObject"/>
        <xs:element name="Control" type="ControlType" substitutionGroup="RootLevelObject"/>
        <xs:element name="Controller" type="ControllerType" substitutionGroup="RootLevelObject"/>
        <xs:element name="Protector" type="ProtectorType" substitutionGroup="RootLevelObject"/>
        <xs:element name="ContentKey" type="ContentKeyType" substitutionGroup="RootLevelObject"/>
        <!-- элементы ключа -->
        <xs:element name="SecretKey" type="KeyType"/>
        <xs:element name="PublicKey" type="PairedKeyType"/>
        <xs:element name="PrivateKey" type="PairedKeyType"/>
        <xs:element name="KeyData" type="KeyDataType"/>
        <!-- other элементы -->
        <xs:element name="AttributeList" type="AttributeListType"/>
        <xs:element name="Attribute" type="AttributeType"/>
        <xs:element name="ExtensionList" type="ExtensionListType"/>
        <xs:element name="Extension" type="ExtensionType" substitutionGroup="RootLevelObject"/>
        <xs:element name="LinkFrom" type="OctopusObjectReferenceType"/>
        <xs:element name="LinkTo" type="OctopusObjectReferenceType"/>
        <xs:element name="Id" type="xs:string"/>
        <xs:element name="Digest" type="DigestType"/>
        <xs:element name="ControlProgram" type="ControlProgramType"/>
        <xs:element name="CodeModule" type="CodeModuleType"/>
        <xs:element name="ControlReference" type="OctopusObjectReferenceType"/>
        <xs:element name="ContentKeyReference" type="OctopusObjectReferenceType"/>
        <xs:element name="ContentReference" type="OctopusObjectReferenceType"/>
        <xs:element name="ProtectedTargets" type="ProtectedTargetsType"/>
        <xs:element name="ControlledTargets" type="ControlledTargetsType"/>
        <!-- scuba -->
        <xs:element name="ScubaKeys" type="ScubaKeysType"/>
        <!-- базовый тип для объектов Octopus -->
        <xs:complexType name="RootLevelObjectType"/>
        <xs:complexType name="OctopusObjectType">
         <xs:complexContent>
          <xs:extension base="RootLevelObjectType">
           <xs:sequence>
            <xs:element ref="AttributeList" minOccurs="0"/>
            <xs:element ref="ExtensionList" minOccurs="0"/>
           </xs:sequence>
           <xs:attribute name="id" type="xs:string" use="optional"/>
          </xs:extension>
         </xs:complexContent>
        </xs:complexType>
        <xs:complexType name="AnyContainerType">
         <xs:complexContent>
          <xs:extension base="RootLevelObjectType">
           <xs:sequence>
            <xs:any processContents="lax"/>
           </xs:sequence>
          </xs:extension>
         </xs:complexContent>
        </xs:complexType>
        <xs:complexType name="ExtensionType">
         <xs:complexContent>
```

```
    <xs:extension base="AnyContainerType">
     <xs:sequence minOccurs="0">
      <xs:element ref="Digest" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="id" type="xs:string" use="required"/>
     <xs:attribute name="subject" type="xs:string"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="ExtensionListType">
   <xs:sequence>
    <xs:element ref="Extension" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AttributeListType">
   <xs:sequence>
    <xs:element ref="Attribute" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AttributeType">
   <xs:simpleContent>
    <xs:extension base="xs:string">
     <xs:attribute name="name" type="xs:string" use="required"/>
     <xs:attribute name="type" type="xs:string" default="string"/>
    </xs:extension>
   </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="DigestType">
   <xs:sequence>
    <xs:element ref="ds:DigestMethod"/>
    <xs:element ref="ds:DigestValue"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="OctopusObjectReferenceType">
   <xs:sequence>
    <xs:element ref="Id"/>
    <xs:element ref="Digest" minOccurs="0"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ProtectedTargetsType">
   <xs:sequence>
    <xs:element ref="ContentReference" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="ControlledTargetsType">
   <xs:sequence>
    <xs:element ref="ContentKeyReference" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
  <!-- Тип Bundle -->
  <xs:complexType name="BundleType">
   <xs:sequence>
    <xs:element ref="RootLevelObject" maxOccurs="unbounded"/>
    <xs:element ref="ds:Signature" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
  <!-- Типы Node -->
  <xs:complexType name="NodeType">
   <xs:complexContent>
    <xs:extension base="OctopusObjectType"/>
   </xs:complexContent>
  </xs:complexType>
  <!-- Типы Link -->
```

```
<xs:complexType name="LinkType">
 <xs:complexContent>
  <xs:extension base="OctopusObjectType">
   <xs:sequence>
    <xs:element ref="LinkFrom"/>
    <xs:element ref="LinkTo"/>
    <xs:element ref="Control" minOccurs="0"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<!-- Типы Protector -->
<xs:complexType name="ProtectorType">
 <xs:complexContent>
  <xs:extension base="OctopusObjectType">
   <xs:sequence>
    <xs:element ref="ContentKeyReference"/>
    <xs:element ref="ProtectedTargets"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<!-- Типы Control -->
<xs:complexType name="CodeModuleType">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="байт-кодType" use="required"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="ControlProgramType">
 <xs:sequence>
  <xs:element ref="CodeModule"/>
 </xs:sequence>
 <xs:attribute name="type" use="required"/>
</xs:complexType>
<xs:complexType name="ControlType">
 <xs:complexContent>
  <xs:extension base="OctopusObjectType">
   <xs:sequence>
    <xs:element ref="ControlProgram"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<!-- Controller Type -->
<xs:complexType name="ControllerType">
 <xs:complexContent>
  <xs:extension base="OctopusObjectType">
   <xs:sequence>
    <xs:element ref="ControlReference"/>
    <xs:element ref="ControlledTargets"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<!-- Типы Key -->
<xs:complexType name="KeyType">
 <xs:sequence>
  <xs:element ref="KeyData"/>
 </xs:sequence>
 <xs:attribute name="id" type="xs:string" use="required"/>
 <xs:attribute name="usage" type="xs:string" use="необязательный"/>
```

```
</xs:complexType>
<xs:complexType name="PairedKeyType">
 <xs:complexContent>
  <xs:extension base="KeyType">
   <xs:attribute name="pair" type="xs:string" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<xs:complexType name="KeyDataType" mixed="true">
 <xs:sequence>
  <xs:element ref="xenc:EncryptedData" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute name="encoding" use="required">
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:enumeration value="xmlenc"/>
    <xs:enumeration value="base64"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:attribute>
 <xs:attribute name="format" use="required">
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:enumeration value="PKCS#8"/>
    <xs:enumeration value="X.509"/>
    <xs:enumeration value="RAW"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:attribute>
</xs:complexType>
<!-- Типы ContentKey -->
<xs:complexType name="ContentKeyType">
 <xs:complexContent>
  <xs:extension base="OctopusObjectType">
   <xs:sequence>
    <xs:element ref="SecretKey"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
<!-- Расширения Scuba -->
<xs:complexType name="ScubaKeysType">
 <xs:sequence>
  <xs:element ref="SecretKey" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element ref="PublicKey" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element ref="PrivateKey" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
</xs:schema>
```

Иллюстративная схема, зависящая от приложения:

```
<?версия xml="1.0" кодировка="UTF-8"?>
<xs:schema targetNamespace="http://intertrust.com/kformat/1.0"
xmlns="http://intertrust.com/kformat/1.0" xmlns:oct="http://intertrust.com/Octopus/1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<!-- импорты -->
<xs:import namespace="http://intertrust.com/Octopus/1.0" schemaLocation="Octopus.xsd"/>
<!-- элементы -->
<xs:element name="Torpedo" type="TorpedoType"/>
<xs:element name="BroadcastKey" type="BroadcastKeyType"/>
```

```
<xs:element name="BroadcastKeyMethod" type="BroadcastKeyMethodType"/>
<!-- типы -->
<xs:complexType name="TorpedoType">
 <xs:sequence>
  <xs:element ref="BroadcastKey"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="BroadcastKeyType">
 <xs:sequence>
  <xs:element ref="BroadcastKeyMethod"/>
  <xs:element ref="oct:KeyData"/>
 </xs:sequence>
 <!-- id является именем MNK -->
 <xs:attribute name="id" type="xs:string"/>
 <!-- источник является именем MKT -->
 <xs:attribute name="source" type="xs:string"/>
</xs:complexType>
<xs:complexType name="BroadcastKeyMethodType">
 <xs:attribute name="Algorithm" fixed="http://marlin-drm.com/mangrove/1.0"/>
</xs:complexType>
</xs:schema>
```

B.1. Дополнительные ограничения.

B.1.1. Узлы.

В одном варианте осуществления заданы следующие типы узлов.

Узел индивидуальности Octopus, которые являются корневыми узлами данного механизма DRM (например, узел «устройство» или узел «программное обеспечение ПК»).

Другие типы узлов, например узлы «пользователь» или узлы для группы пользователей, например, узлы подписки или узлы принадлежности.

В одном варианте осуществления узлы содержат ключи (например, в Extensions, например, ScubaKeys), и должна быть возможность разделять открытую информацию узла (например, id, атрибуты и открытые ключи) и его личные расширения (которые, например, несут секретный и личный ключи). Кроме того, должно существовать по одной подписи на часть (открытую или личную), чтобы можно было экспортировать открытый узел со своей подписью, как есть (например, как параметр запроса к следующей лицензий).

В одном варианте осуществления, личные расширения переносятся в ExternalExtension и подписываются. Открытый узел и его личные расширения могут быть упакованы в одном и том же элементе <Bundle>, или могут приходить раздельно. Пример подписанного узла индивидуальности Octopus приведен ниже в дополнении A к приложению B.

B.1.1.1. Атрибуты.

В одном варианте осуществления, каждая XML-кодировка объекта «узел» несет <AttributeList> со следующими полями <Attribute>:

Для индивидуальностей Octopus:

```
<AttributeList xmlns="http://intertrust.com/Octopus/1.0">
 <Attribute name="urn:x-marlin.intertrust.com:type">...</Attribute>
 <Attribute name="urn:x-marlin.intertrust.com:dnk_id">...</Attribute>
 <Attribute name="urn:x-marlin.intertrust.com:manufacturer">...</Attribute>
 <Attribute name="urn:x-marlin.intertrust.com:model">...</Attribute>
 <Attribute name="urn:x-marlin.intertrust.com:version">...</Attribute>
</AttributeList>
```

Для узлов других типов:

```
<AttributeList xmlns="http://intertrust.com/Octopus/1.0">
 <Attribute name="urn:x-marlin.intertrust.com:type">...</Attribute>
</AttributeList>
```

B.1.1.2 Расширения.

Согласно дополнению A к этому приложению B, в одном варианте осуществления узлы индивидуальности Octopus несут расширения для ScubaKeys (ключи совместного пользования и конфиденциальности) и Torpedo (широковещательный секретный ключ). Узлы других типов несут только ключи совместного пользования Scuba.

Все открытые ключи переносятся внутри элемента <Node> в <Extension> в <ExtensionList>. Другие ключи переносятся в отдельном элементе <Extension> вне элемента <Node>.

В одном варианте осуществления, расширения <ScubaKeys> подписаны в <Node>. В этом варианте осуществления, внутреннее <Extension>, несущее <ScubaKeys> внутри <Node> (открытые ключи) должно включать в себя элемент <ds:DigestMethod>, а также элемент <ds:DigestValue>. Секретные ключи, переносимые во внешнем <Extension>, должны быть подписаны, и это делается путем подписывания

всего расширения. Аналогично, подписано расширение <Torpedo>.

B.1.2. Связи.

В одном варианте осуществления, элементы <LinkTo> и <LinkFrom> элемента <Link> содержат только элемент <Id> и ни одного элемента <Digest>. Элемент <Control> является необязательным. Дополнение C к этому приложению B содержит пример подписанного объекта «связь».

B.1.2.1. Атрибуты.

В одном варианте осуществления, связи не имеют обязательных атрибутов. Это означает, что <AttributeList> не обязателен и будет игнорироваться гибкой реализацией.

B.1.2.2. Расширения.

В иллюстративном варианте осуществления, показанном в этом приложении B, связи имеют внутренние расширения <ScubaKeys>, переносимые внутри <Link>, и, таким образом, элемент <ExtensionList> является обязательным. Кроме того, расширение <ScubaKeys> в связи не подписано, и, таким образом, ни элемент <ds:DigestMethod>, ни элемент <ds:DigestValue> не переносится внутри элемента <Extension>. Это расширение <ScubaKeys> содержит зашифрованную версию открытого/секретного ключей совместного пользования Scuba (в элементах <PrivateKey> и <SecretKey>) "конечного узла" с открытым или секретным ключом совместного пользования Scuba от "начального узла". Это шифрование сигнализируется с использованием синтаксиса шифрования XML. Согласно варианту осуществления, представленному в этом приложении B, атрибут "encoding" элемента <KeyData>, дочернего по отношению к элементам <PrivateKey> и <SecretKey>, задан равным "xmlenc". Дочерним элементом этого элемента <KeyData> будет элемент <xenc:EncryptedData>. Имя ключа шифрования объявлено в элементе <KeyInfo>/<KeyName>.

В одном варианте осуществления, если ключ шифрования является открытым ключом, то элемент <KeyName> является именем пары, которой принадлежит ключ;

если зашифрованные данные, например секретный ключ, слишком велики для шифрования непосредственно открытым ключом, генерируется промежуточный 128-битовый секретный ключ. Затем данные шифруются этим промежуточным ключом с использованием, например, aes-128-cbc, и промежуточный ключ шифруется открытым ключом (с использованием элемента <EncryptedKey>).

Фрагмент XML-кода будет выглядеть примерно так:

```
<!—E(I, data) -->
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
 <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
 <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
  <!-- E(PUBa, I) -->
  <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
   <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
   <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <KeyName>urn:x-octopus.intertrust.com:key-pair:300a</KeyName>
   </KeyInfo>
   <CipherData>
    <CipherValue>
fFeGD4KAPEmESz/jW6CkbRegpM5kyH0Oy/o/uDQ78PaShtvUMoozeO4a0b785YnB
13Qa1ZUEYqR9V5TCUaOcH7wxxvBEIsd1nYKkVOgW/kFnRr98UDFvU90PRqaEP/SA
Bb+JuAUmvxYX47qOVQqBQGGqzFssBDKmUk+s98dkPR8=
    </CipherValue>
   </CipherData>
  </EncryptedKey>
 </KeyInfo>
 <CipherData>
  <CipherValue>
c8LBj4BLzGOYv/GT3Y4w2XcwTYbr8fHNJhCOQjULuvoha/QYvZKKCpUY+nuCXC/s
t9TU+8tMtaMt1GUpkCZQhSaTNcluCSxOyBoA6Xh/bmyZLDJ78+aJ/slTmfNpJGdb
vTaI7x9DD1Mp1mvFEjpAUjTTvruN32g4bxsF7FD8C1RWNAc4hS96nFDgrmzoO5pR
dda6mswFKG5B0kY7mYbhacb1owXkAk1Wc/OuXA+QLHdUthxeajoXNPfAGRz9FM3b
puJxbxDAaaAJDxoReiTtS1nGaHhqa1hvLCpKk1zHBowHyvTvDLElLjHYEPeG6xSH
BbzpT298tdKUhXfaY6vvdceMdVXuBVL3eZP1jkJHDxeaBy1ce8xlQKZpo6Pjuxlb
bn5KUMt/PxWp7rLa5s786S740cwuN63+ZRgienxPK1CnYO3htMJ7hh/agvO9IyUD
RvcgnSEY9KA5Exy/6gIS/gouljFU8r7056XcE4/IBodTWDkfyli/y8q5QA/0VaD9
Y3oER1p3pYuHwn/IeXM4gsBD31cgd7nvfK7lKYkZjowR9P6pSy57a+K4LZKDmfUH
zG/gZs2XcoPb9o6mVAEEej7+aLwqmoileykkR+0pkFntvqvXYRPkphhcVdzjzlMV
scpXBXfWx7wbQURXkiew7R4RihQy3wcv+ZFJpl9NsAE1yqyWy4rBobzZ7cTNMtfR
znhVlt+Wwq5G0IBxzU9WIFzFd/Rn2H9L4TI71LCa4VR3uNpf+XM8lp9LjLPRUnNh
28KrMdAddceyopYyiIF5p8idfh0//a/LKdE7JAk0q9ewk19ryqfl6CFeKI5oOMjh
kzNx3BR/iHxm31HIe3ZKtA==</CipherValue>
</CipherData>
</EncryptedData>
```

B.1.3. Объекты «лицензия».

В дополнении C к этому приложению B приведен пример подписанной лицензии (до того, как произойдет первая отмена, см. ниже раздел ContentKey).

B.1.3.1. Protector.

В иллюстративном варианте осуществления, показанном в этом приложении B, элемент <ContentKeyReference> и элементы <ContentReference> (например, внутри элемента <ProtectedTargets>) содержат только element <Id> и ни одного элемента <Digest>. В этом иллюстративном варианте осуществления, объекты Protector не содержат обязательных атрибутов или расширений; элементы <AttributeList> и <ExtensionList> являются необязательными и будут игнорироваться.

B.1.3.2. ContentKey.

В иллюстративном варианте осуществления, показанном в этом приложении B, объекты ContentKey не содержат обязательных атрибутов или расширений. Поэтому, элементы <AttributeList> и <ExtensionList> являются необязательными и будут игнорироваться.

В одном варианте осуществления, элементы <ContentKey> содержат элемент <SecretKey>, который представляет фактический ключ, который будет использоваться для дешифрования контента. <KeyData>, связанный с <SecretKey>, зашифрован. В одном варианте осуществления, обязательно, чтобы атрибут "encoding" элемента <KeyData> был задан равным "xmlenc".

В одном варианте осуществления, существует два разных варианта для объектов ContentKey: (1) До первой отмены устройства или приложения ПК: в этом случае, ключ контента Кс, представленный элементом <SecretKey>, шифруется только ключом Scuba (открытым или секретным) сущности, к которой привязан контент (например, пользователя). (2) После первой отмены, когда ключ контента шифруется согласно схеме широковещательного шифрования Mangrove. Затем результирующие данные шифруются ключом Scuba (открытым или секретным) сущности, к которой привязан контент. В этом случае, мы имеем супер-шифрование.

Иллюстративные методы шифрования элемента <EncryptedData> в случае супер-шифрования описаны здесь в другом месте. Ниже объясняется, как применить это к случаю b.

В одном варианте осуществления, синтаксис xmlenc для шифрования ключа контента Кс согласно схеме широковещательного шифрования Mangrove таков:

```
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
 <EncryptionMethod Algorithm="see (*)"/>
 <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
  <KeyName>see (**)</KeyName>
 </KeyInfo>
 <CipherData>
  <CipherValue>see (***)</CipherValue>
 </CipherData>
</EncryptedData>
```

(*) это URL, идентифицирующий схему широковещательного шифрования Mangrove, которая, в одном варианте осуществления, также является алгоритмом <BroadcastKeyMethod> расширения <Torpedo> в вызове схемы xml, зависящей от приложения "kformat.xsd".

(**) это имя дерева ключей Mangrove. В одном варианте осуществления, это значение должно быть таким же, как у атрибута «source» элемента <BroadcastKey>, заданного в kformat.xsd.

(***) это значение, в кодировке base64, последовательности ASN.1, представляющей шифрование ключа контента Кс согласно алгоритму широковещательных ключей Mangrove:

```
SEQUENCE {
    tags BIT STRING
    keys OCTET STRING
}
```

В одном варианте осуществления последовательность байтов <EncryptedData>, согласно вышесказанному, шифруется ключом совместного пользования scuba (открытым или секретным) сущности, к которой привязана лицензия. Если используется открытый ключ, то применяются те же соглашения, которые описаны ниже (см., например, шифрование открытым ключом), и необходим промежуточный ключ, если последовательность байтов <EncryptedData> слишком велика для открытого ключа RSA 1024. Пример XML-кодирования такого объекта ContentKey можно найти в дополнении D к этому приложению B.

B. 1.3.3. Controller.

В одном варианте осуществления объекты «контроллер» не содержат обязательных атрибутов или расширений. Поэтому элементы <AttributeList> и <ExtensionList> являются необязательными и будут игнорироваться гибкой реализацией.

В одном варианте осуществления значение атрибута Algorithm элементов <DigestMethod> всегда равны http://www.w3.org/2000/09/xmldsig#sha1.

В одном варианте осуществления <ControlReference> должен иметь элемент <Digest>. Элемент

<DigestValue> должен содержать кодировку base64 дайджеста объекта управления, на который имеется ссылка.

В одном варианте осуществления, если подпись на контроллере является подписью PKI (rsa-sha1), элементы <ContentKeyRefence> (в элементах <ControlledTargets>) должны включать в себя элемент <Digest>, и элемент <DigestValue> должен содержать дайджест незашифрованного ключа контента, внедренного в объект ContentKey.

B.1.3.4. Control.

В одном варианте осуществления, объекты управления не содержат обязательных атрибутов или расширений. Поэтому элементы <AttributeList> и <ExtensionList> являются необязательными и будут игнорироваться гибкой реализацией.

В одном варианте осуществления, атрибут «type» элемента <ControlProgram> задан равным "plankton", и атрибут byte-codeType элемента <CodeModule> задан равным "Plankton-1-0."

Приложение B

Дополнение A. Пример подписанного узла индивидуальности Octopus

```
<Bundle xmlns:ds=http://www.w3.org/2000/09/xmldsig#
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" xmlns=http://intertrust.com/Octopus/1.0
xmlns:xsi=http://www.w3.org/2001/XMLSchema-экземпляр
xsi:schemaLocation="http://intertrust.com/kformat/1.0
C:\DOCUME~1\julien\Desktop\kformat\kformat.xsd">
 <!-- сначала узел с открытой информацией-->
 <Node id="urn:kformat:device:0001">
  <AttributeList>
   <Attribute name="urn:x-marlin.intertrust.com:type">device</Attribute>
   <Attribute name="urn:x-marlin.intertrust.com:dnk_id">urn:kformat:mangrove:0001</Attribute>
   <Attribute name="urn:x-marlin.intertrust.com:manufacturer_id">SONY</Attribute>
   <Attribute name="urn:x-marlin.intertrust.com:model">urn:sony:walkman</Attribute>
   <Attribute name="urn:x-marlin.intertrust.com:version">urn:sony:walkman:002a</Attribute>
  </AttributeList>
  <ExtensionList>
   <Extension id="urn:kformat:device:0001:scuba:public">
    <ScubaKeys>
     <PublicKey id="urn:kformat:device:0001:scuba:public:sharing"
          pair="urn:kformat:device:0001:scuba:pair:sharing">
      <KeyData encoding="base64" format="X509">MIIC…MEbB</KeyData>
     </PublicKey>
     <PublicKey id="urn:kformat:device:0001:scuba:public:confidentiality"
          usage="confidentiality"
          pair="urn:kformat:device:0001:scuba:pair:confidentiality">
      <KeyData encoding="base64" format="X.509">MIIChDCC… vh8BM52</KeyData>
     </PublicKey>
    </ScubaKeys>
    <Digest>
     <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
```

```
                Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue xmlns="http://www.w3.org/2000/09/xmldsig#">OGZGBY8OpQXs</DigestValue>
     </Digest>
    </Extension>
   </ExtensionList>
  </Node>
  <!-- затем личное расширение Scuba -->
  <Extension id="urn:kformat:device:0001:scuba:private" subject="urn:kformat:device:0001">
   <ScubaKeys>
    <PrivateKey id="urn:kformat:device:0001:scuba:private:sharing"
          pair="urn:kformat:device:0001:scuba:pair:sharing">
     <KeyData encoding="base64" format="PKCS8">MIICdgIBADAN... DXywQLg==</KeyData>
    </PrivateKey>
    <PrivateKey id="urn:kformat:device:0001:scuba:private:confidentiality"
          usage="confidentiality"
          pair="urn:kformat:device:0001:scuba:pair:confidentiality">
     <KeyData encoding="base64" format="PKCS8">MIICdwIBADAN... q4olog34=</KeyData>
    </PrivateKey>
    <SecretKey id="urn:kformat:device:0001:scuba:secret:sharing">
     <KeyData encoding="base64" format="RAW">Z1n2/2cbz1oO/fZo9xtmyA==</KeyData>
    </SecretKey>
    <SecretKey id="urn:kformat:device:0001:scuba:secret:confidentiality"
         usage="confidentiality">
     <KeyData encoding="base64" format="RAW">0CJ8bcORW6GLX4GzT7XKvg==</KeyData>
    </SecretKey>
   </ScubaKeys>
  </Extension>
  <!-- затем личное расширение Torpedo -->
  <Extension id="urn:kformat:device:0001:torpedo" subject="urn:kformat:device:0001">
   <Torpedo xmlns="http://intertrust.com/kformat/1.0">
    <BroadcastKey id="urn:kformat:mangrove:0001">
     <BroadcastKeyMethod Algorithm="http://marlin-drm.com/mangrove/1.0"/>
     <KeyData              xmlns="http://intertrust.com/Octopus/1.0"              encoding="base64"
format="RAW">....</KeyData>
    </BroadcastKey>
   </Torpedo>
  </Extension>
  <!-- затем подпись на открытой части -->
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="urn:kformat:device:0001">
     <Transforms>
      <Transform Algorithm="http://www.octopus-drm.com/2004/07/format-independent-cano#"/>
     </Transforms>
     <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
     <DigestValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</DigestValue>
    </Reference>
   </SignedInfo>
   <SignatureValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</SignatureValue>
   <KeyInfo>
    <X509Data>
     <!-- поместить сертификат открытого ключа для ключа подписи здесь -->
     <X509Certificate>...</X509Certificate>
     <!-- и цепь сертификатов без корня, если необходимо -->
     <X509Certificate>...</X509Certificate>
    </X509Data>
   </KeyInfo>
  </Signature>
  <!-- затем подпись на личной части -->
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
```

```
    <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="urn:kformat:0001:scuba:private">
     <Transforms>
      <Transform Algorithm="http://www.octopus-drm.com/2004/07/format-independent-cano#"/>
     </Transforms>
     <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
     <DigestValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</DigestValue>
    </Reference>
    <Reference URI="urn:kformat:device:0001:torpedo">
     <Transforms>
      <Transform Algorithm="http://www.octopus-drm.com/2004/07/format-independent-cano#"/>
     </Transforms>
     <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
     <ds:DigestValue>97mDfnw0vF/ECQHcvDk</ds:DigestValue>
    </Reference>
   </SignedInfo>
   <SignatureValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</SignatureValue>
   <KeyInfo>
    <X509Data>
     <!-- поместить сертификат открытого ключа для ключа подписи здесь -->
     <X509Certificate>...</X509Certificate>
     <!-- и цепь сертификатов без корня, если необходимо -->
     <X509Certificate>...</X509Certificate>
    </X509Data>
   </KeyInfo>
  </Signature>
  </Bundle>
```

Приложение В. Дополнение В. Пример подписанной связи Octopus

```
<?версия xml="1.0" кодировка="UTF-8"?>
<!--Образец файла XML, сгенерированного XMLSPY v2004 rel. 2 U (http://www.xmlspy.com)-->
<Bundle      xmlns="http://intertrust.com/Octopus/1.0"      xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
экземпляр" xsi:schemaLocation="http://intertrust.com/Octopus/1.0
C:\ws\Octopus\Source\Xml\Schemas\Octopus.xsd">
 <Link id="urn:kformat:link:device:0001:to:user:1234">
  <ExtensionList>
   <Extension id="urn:kformat:link:device:0001:to:user:1234:scuba">
    <ScubaKeys>
     <!-- E(PUBdevice, PRIVuser) -->
     <PrivateKey id="urn:kformat:user:1234:scuba:private:sharing"
           pair="urn:kformat:user:1234:scuba:pair:sharing">
      <KeyData encoding="xmlenc" format="PKCS8">
       <!-- E(I, PRIVuser) I: промежуточный ключ-->
       <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
         <!-- E(PUBdevice, I) -->
         <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
           <KeyName>urn:kformat:device:0001:scuba:pair:sharing</KeyName>
          </KeyInfo>
          <CipherData>
           <CipherValue>fFeGD4K... s98dkPR8=</CipherValue>
          </CipherData>
         </EncryptedKey>
        </KeyInfo>
        <CipherData>
```

```
            <CipherValue>
c8LBj4BLzGOYv...HIe3ZKtA==</CipherValue>
           </CipherData>
          </EncryptedData>
         </KeyData>
        </PrivateKey>
        <!-- E(PUBdevice, Suser) -->
        <SecretKey id="urn:kformat:user:1234:secret:sharing">
         <KeyData encoding="xmlenc" format="RAW">
          <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
           <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
           <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>urn:kformat:device:0001:scuba:pair:sharing</KeyName>
           </KeyInfo>
           <CipherData>
            <CipherValue>OHVaH... kjLA=</CipherValue>
           </CipherData>
          </EncryptedData>
         </KeyData>
        </SecretKey>
       </ScubaKeys>
      </Extension>
     </ExtensionList>
     <LinkFrom>
      <Id>urn:kformat:device:0001</Id>
     </LinkFrom>
     <LinkTo>
      <Id>urn:kformat:user:1234</Id>
     </LinkTo>
    </Link>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
     <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="urn:kformat:link:device:0001:to:user:1234">
       <Transforms>
        <Transform Algorithm="http://www.octopus-drm.com/2004/07/format-independent-cano#"/>
       </Transforms>
       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <DigestValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</DigestValue>
      </Reference>
     </SignedInfo>
     <SignatureValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</SignatureValue>
     <KeyInfo>
      <X509Data>
       <!-- поместить сертификат открытого ключа для ключа подписи здесь -->
       <X509Certificate>...</X509Certificate>
       <!-- и цепь сертификатов без корня, если необходимо -->
       <X509Certificate>...</X509Certificate>
      </X509Data>
     </KeyInfo>
    </Signature>
    </Bundle>
```

Приложение В. Дополнение С. Пример подписанной лицензии Octopus (без отмены)

```
<Bundle xmlns="http://intertrust.com/Octopus/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi.schemaLocation="http://intertrust.com/Octopus/1.0
C:\ws\Octopus\Source\Xml\Schemas\Octopus.xsd">
 <ContentKey id="urn:x-octopus.intertrust.com:контент-key:2002">
  <SecretKey id="urn:x-octopus.intertrust.com:secret-key:2002">
   <KeyData encoding="xmlenc" format="RAW">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
     <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
     <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <KeyName>urn:x-octopus.intertrust.com:secret-key:303c</KeyName>
     </KeyInfo>
     <CipherData>
      <CipherValue>
MCR0LGaoyuO2o6zsIW9IrOOSMfhuZCtV20o94/OfQ5dHblJ3q2vZrgwRbJepLvRa
      </CipherValue>
     </CipherData>
    </EncryptedData>
   </KeyData>
  </SecretKey>
 </ContentKey>
 <ContentKey id="urn:x-octopus.intertrust.com:контент-key:2001">
  <SecretKey id="urn:x-octopus.intertrust.com:secret-key:2001">
   <KeyData encoding="xmlenc" format="RAW">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
     <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
     <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <KeyName>urn:x-octopus.intertrust.com:key-pair:300c</KeyName>
     </KeyInfo>
     <CipherData>
      <CipherValue>
LD51cJ71Bswwb2GttPoPjMytFn3ooeI7vhZPA5mKY06R82KZjxFDtcCmbOIYZ5Hv
6ldqQ3hy74/mQF3AJ1jRXa9/ymmasVBxsJnv426B9/JkzTT4CGqNjS+WPOKL9NZC
qnRWguJmk8dQ+jaxW51SQSjp4MCpGZB63zfvcuBD7qE=
      </CipherValue>
     </CipherData>
    </EncryptedData>
   </KeyData>
  </SecretKey>
 </ContentKey>
 <Control id="urn:x-octopus.intertrust.com:control:0001">
  <ControlProgram type="Plankton">
   <CodeModule байт-кодType="Plankton-1-0">
AAABUnBrQ00AAAA2cGtFWAAAAAIOR2xvYmFsLk9uTG9hZAAAAAAAAEkFjdGlvbi5QbGF5LkNoZ
WNrAAAAAFgAAACmcGtDUwEAAAAEGgEAAAAABQEAAAACIAMBAAAABBoBAAAAHgUbAQ
AAACwYAQAAAAQaAQAAACIFAQAAAAIgAwEAAAAEGgEAAAA7BRsBAAAABhgBAAAAABU
B/////xUBAAAABBoBAAAAPwUBAAAABBoBAAAAHgUaIAEAAAAgGAEAAAAEGgEAAAA7BRog
AQEOX3oLAQAAAAYYAQAAAAAVAf////8VAAAAbnBrRFNPY3RvcHVzLkxpbmtzLklzTm9kZVJlY
WNoYWJsZQAAAAAU3lzdGVtLkhvc3QuR2V0VGltZVN0YW1wAAAAAAB1cm46eC1vY3RvcHVzL
mludGVydHJ1c3QuY29tOm5vZGU6MDAwMwA=
   </CodeModule>
  </ControlProgram>
 </Control>
 <Protector>
  <ContentKeyReference>
   <Id>urn:x-octopus.intertrust.com:контент-key:2002</Id>
  </ContentKeyReference>
  <ProtectedTargets>
   <ContentReference>
    <Id>urn:x-octopus.intertrust.com:контент:2001</Id>
   </ContentReference>
   <ContentReference>
    <Id>urn:x-octopus.intertrust.com:контент:2002</Id>
   </ContentReference>
  </ProtectedTargets>
 </Protector>
 <Protector>
  <ContentKeyReference>
   <Id>urn:x-octopus.intertrust.com:контент-key:2001</Id>
```

```
    </ContentKeyReference>
    <ProtectedTargets>
     <ContentReference>
      <Id>urn:x-octopus.intertrust.com:контент:2003</Id>
     </ContentReference>
     <ContentReference>
      <Id>urn:x-octopus.intertrust.com:контент:2004</Id>
     </ContentReference>
    </ProtectedTargets>
   </Protector>
   <Controller id="urn:x-octopus.intertrust.com:controller:0001">
    <ControlReference>
     <Id>urn:x-octopus.intertrust.com:control:0001</Id>
     <Digest>
      <DigestMethod Algorithm=http://www.w3.org/2000/09/xmldsig#sha1
xmlns="    p://www.w3.org/2000/09/xmldsig#"/>
      <DigestValue
xmlns="http://www.w3.org/2000/09/xmldsig#">02ACF5674287FF45CFA5A66D70125FF5601A63F7</Dig
estValue>
     </Digest>
    </ControlReference>
    <ControlledTargets>
     <ContentKeyReference>
      <Id>urn:x-octopus.intertrust.com:контент-key:2002</Id>
     </ContentKeyReference>
     <ContentKeyReference>
      <Id>urn:x-octopus.intertrust.com:контент-key:2001</Id>
     </ContentKeyReference>
    </ControlledTargets>
   </Controller>
   <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
     <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
     <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
     <Reference URI="urn:x-octopus.intertrust.com:controller:0001">
      <Transforms>
       <Transform Algorithm="http://www.octopus-drm.com/2004/07/format-independent-cano#"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>A42CZFK4DQvb/M0wqOLZRnyiS1Y=</DigestValue>
     </Reference>
    </SignedInfo>
    <SignatureValue>gI5QoD7MUAgjcpkPiciZhoSHbEQ=</SignatureValue>
    <KeyInfo>
     <KeyName>urn:x-octopus.intertrust.com:secret-key:2002;urn:x-octopus.intertrust.com:secret-
key:2001</KeyName>
    </KeyInfo>
   </Signature>
</Bundle>
```

Приложение B. Дополнение D. Пример ContentKey с отменой

```
<ContentKey id="urn:x-octopus.intertrust.com:content-key:2001">
 <SecretKey id="urn:x-octopus.intertrust.com:secret-key:2001">
  <KeyData encoding="xmlenc" format="RAW">
   <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#">
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
     <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
       <KeyName>urn:kformat:user:0001:scuba:pair:sharing</KeyName>
      </KeyInfo>
```

```
    <CipherData>
      <CipherValue>E(PUBuser, I)</CipherValue>
    </CipherData>
  </EncryptedKey>
</KeyInfo>
<CipherData>
 <CipherValue>
 Шифрование элемента EncryptedData, содержащее
 шифрование Kc согласно схеме широковещательного шифрования
 (см. примечание по xmlenc и широковещательному шифрованию ключей
 в разделе ContentKey) с помощью промежуточного ключа I.
 </CipherValue>
</CipherData>
</EncryptedData>
</KeyData>
</SecretKey>
</ContentKey>
```

Приложение C

В этом приложении C приведен пример простого профиля для использования с вышеописанным протоколом автозагрузки. Кроме того, предусмотрены простая каноническая сериализация, иллюстративная маршализация XML, и иллюстративный WSDL для Octopus Bootstrap SOAP Web Serivce.

Простой профиль

В одном варианте осуществления, используется простой профиль, имеющий следующий состав:

Имя профиля	Простой профиль
Алгоритм шифрования открытым ключом	http://www.w3.org/2001/04/xmlenc#rsa-1_5
Алгоритм подписи открытым ключом	http://www.w3.org/2000/09/xmldsig#rsa-sha1
Алгоритм шифрования секретным ключом	http://www.w3.org/2001/04/xmlenc#aes128-cbc
Алгоритм подписи секретным ключом	http://www.w3.org/2000/09/xmldsig#hmac-sha1
Алгоритм дайджеста	http://www.w3.org/2000/09/xmldsig#sha1
Формат сертификата	X.509 (версия 3)
Маршализация сообщений	Простая маршализация XML 1.0
Минимальный размер нонса	16 байтов
Каноническая сериализация объекта	Простая каноническая сериализация 1.0

Простая каноническая сериализация 1.0.

В одном варианте осуществления вышеописанная простая каноническая последовательность байтов, используемая в простом профиле, состоит в построении последовательностей байтов из значений полей объектов в сообщениях. Каждое сообщение и каждый объект состоит из одного или нескольких полей. Каждое поле является либо простым полем, либо составным полем.

Простые поля могут быть четырех типов: целое число, строка, последовательность байтов или массив полей. Сложные поля состоят из одного или нескольких подполей, каждое из которых может быть простым или составным.

В одном варианте осуществления существуют следующие правила построения канонической последовательности байтов для каждого типа поля:

Сложные поля

Поле 0	Поле 1	Поле 2	...

Каноническая последовательность байтов является сцепкой канонических последовательностей байтов каждого из подполей (необязательные поля не пропускаются, но сериализуются согласно правилу для необязательных полей).

Массив полей

Счетчик полей	Поле 0	Поле 1	...

Счетчик полей закодирован в виде последовательности из 4 байтов в обратном порядке следования байтов, после него следуют канонические последовательности байтов каждого из полей. Если счетчик полей равен 0, то после 4-байтового счетчика полей ничего нет (в этом случае, все 4 байта имеют значение 0).

Целое число

I0	I1	I2	I3

32-битовое знаковое значение, закодированное в виде последовательности из 4 байтов, в обратном порядке следования байтов.

Строка

Счетчик байтов	байт 0	байт 1	...

Строка представлена последовательностью 8-битовых байтов в кодировке UTF-8. Счетчик байтов закодированной последовательности байтов кодируется в виде последовательности из 4 байтов в обратном порядке следования байтов. После счетчика байтов следует последовательность байтов строки в кодировке UTF-8.

Последовательность байтов

Счетчик байтов	байт 0	байт 1	...

Счетчик байтов закодирован в виде последовательности из 4 байтов в обратном порядке следования байтов (если последовательность байтов пуста, или соответствующее поле пропущено, счетчик байтов равен 0, и после 4-байтового счетчика байтов нет никаких байтовых значений). Каждый байт кодируется как есть.

Простая маршализация XML 1.0

Схема SimpleBootProtocol.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<xs:element name="BootstrapRequestMessage">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="BootstrapRequest"/>
			</xs:sequence>
			<xs:attribute name="Protocol" type="xs:string" use="required"/>
			<xs:attribute name="Profile" type="xs:string" use="required"/>
			<xs:attribute name="Version" type="xs:decimal" use="required"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="BootstrapRequest">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="SessionId"/>
				<xs:element ref="TrustDomain" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="ChallengeRequestMessage">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="ChallengeRequest"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="ChallengeRequest">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="Challenge"/>
				<xs:element ref="Signature"/>
				<xs:element ref="CertificateChain"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="ChallengeResponseMessage">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="SessionKey"/>
				<xs:element ref="EncryptedChallengeResponse"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="EncryptedChallengeResponse" type="xs:base64Binary"/>
	<xs:element name="ChallengeResponse">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="ClientInfo"/>
				<xs:element ref="Challenge"/>
				<xs:element ref="SessionKey"/>
				<xs:element ref="Signature"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="Challenge">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="Cookie"/>
				<xs:element ref="Nonce"/>
				<xs:element ref="SessionId"/>
				<xs:element ref="EncryptionKey" minOccurs="0"/>
			</xs:sequence>
		</xs:complexType>
```

```
        </xs:element>
        <xs:element name="BootstrapResponseMessage">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="EncryptedBootstrapResponse"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="EncryptedBootstrapResponse" type="xs:base64Binary"/>
        <xs:element name="BootstrapResponse">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="SessionId"/>
                    <xs:element ref="Data"/>
                    <xs:element ref="Signature"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="ErrorResponseMessage">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="ErrorResponse"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="ErrorResponse" type="xs:string"/>
        <xs:element name="CertificateChain">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="Certificate" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="TrustDomain" type="xs:string" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="Certificate" type="xs:base64Binary"/>
        <xs:element name="ClientInfo">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="Attribute"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="Attribute" type="xs:string"/>
        <xs:element name="Cookie" type="xs:base64Binary"/>
        <xs:element name="Data" type="xs:base64Binary"/>
        <xs:element name="EncryptionKey" type="xs:base64Binary"/>
        <xs:element name="Nonce" type="xs:base64Binary"/>
        <xs:element name="SessionId" type="xs:string"/>
        <xs:element name="SessionKey" type="xs:base64Binary"/>
        <xs:element name="Signature" type="xs:base64Binary"/>
        <xs:element name="TrustDomain" type="xs:string"/>
</xs:schema>
```

Пример:

```
<BootstrapRequestMessage Protocol="OctopusSimpleBoot" Profile="SimpleProfile" Version="1.0">
 <BootstrapRequest>
  <SessionId>some-unique-session-id-0008</SessionId>
  <TrustDomain>urn:x-octopus.intertrust.com:scuba:boot:trust-domain:test001</TrustDomain>
 </BootstrapRequest>
</BootstrapRequestMessage>

<ChallengeRequestMessage>
 <ChallengeRequest>
  <Challenge>
```

```
    <Cookie>c29tZS11bmlxdWUtc2Vzc2lvbi1pZC0wMDA4</Cookie>
    <Nonce>Mv5VIv73cxo5b+gisQJP8Q==</Nonce>
    <SessionId>some-unique-session-id-0008</SessionId>
    <EncryptionKey>
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCpMY4wvgTJvVPTufNVbdIfTUwOi4FZPtzi3ezetY
9gx51O6dfRn+LKPq1nJsSXCR5ZIvRUyoNZC0Qc3SLobUhXD6uTsrV5xtRKOSxZTLt5DZ15AtddSrAAfF9ba
DGMi5KQP9w7qB2Ci/MmYha4Jix1iUltv0zWIKmSpytgHC8i/QIDAQAB
    </EncryptionKey>
   </Challenge>
   <Signature>
GsWP3yPT36r3e1jZfulUS7xp5w2ei7iTsAJ/YD13fX+pSJrpeKAtq2BTzHQ1AclOorPJwzWHDanc
cui9/rinlg3Drw52bQXLzhZbZLXadlGFP3YP1gTKPuazUCYCLAjYTJbdulWlnTKDtmf34/66H0sz
DCCyxQsdFZhSNk6pyQE=
   </Signature>
   <CertificateChain TrustDomain="urn:x-octopus.intertrust.com:scuba:boot:trust-domain:test001">
    <Certificate>
                MIID...<!-- конечный сертификат сущности -->
    </Certificate>
                MIID...<!-- промежуточный сертификат -->
    <Certificate>
                MIIE...<!-- промежуточный сертификат -->
    </Certificate>
    <Certificate>
MIID...<!-- сертифицирут эти цепи непосредственно доверенному анкеру -->
    </Certificate>
   </CertificateChain>
  </ChallengeRequest>
</ChallengeRequestMessage>

<ChallengeResponseMessage>
  <SessionKey>
PtzJcFT2s1sW7oRZ1a+HASdRmZer4pk4QArFZWYlkUWZclZTN2g2YeCQwORq2J9QXOksU6utKmOmgfE
HY151UdcMFake3CwquvVN6w/7mFH0qtDoc+GhuKe9eQXN2RHa3SlhfR5ShF2A/cwZHd4Nknt4w8MWMD
Dn3SUDd6aS/ZI=
  </SessionKey>
  <EncryptedChallengeResponse>
mQCkPL560D00o...
  </EncryptedChallengeResponse>
</ChallengeResponseMessage>

<ChallengeResponse>
  <ClientInfo>
   <Attribute Name="SomeAttribute">Bla Bla</Attribute>
  </ClientInfo>
  <Challenge>
   <Cookie>c29tZS11bmlxdWUtc2Vzc2lvbi1pZC0wMDA4</Cookie>
   <Nonce>Mv5VIv73cxo5b+gisQJP8Q==</Nonce>
   <SessionId>some-unique-session-id-0008</SessionId>
  </Challenge>
  <SessionKey>bbBG8JsGaApFdNJq6hFrIQ==</SessionKey>
  <Signature>WYMULPpF4lOJ6MiAxd1lueN7p/4=</Signature>
</ChallengeResponse>

<BootstrapResponseMessage>
  <EncryptedBootstrapResponse>
chXTp20+yI7/i1pHLawFOLXdGb...
  </EncryptedBootstrapResponse>
</BootstrapResponseMessage>

<BootstrapResponse>
  <SessionId>some-unique-session-id-0008</SessionId>
  <Data>
PD94bWwgdmVyc...
```

```
 </Data>
 <Signature>
XqCeVRb4YaYAK9Ilj60B5R1hQ03tFpHPw3wMMATbeUfqCpEXfAB7u2/qnjs9jLgWTOOvLDE5C5aVVMvz
lnRnDv0GHLls6g43HusVx7fpazwHoFrb3M3eKwXMoYsI6xpdYy2BX1bs5QT2xdwBv2ClBjo7
KzQfmb/3bYEO+xGdg48=
 </Signature>
</BootstrapResponse>

<ErrorResponseMessage>
 <ErrorResponse Code="6">Some Error Info</ErrorResponse>
</ErrorResponseMessage>
```

WSDL для Bootstrap SOAP Web Service

```
<?версия xml ="1.0" кодировка="UTF-8"?>
<!–
 Этот файл wsdl описывает интерфейс для многоциклового протокола автозагрузки без определения
состояний

 Протокол работает следующим образом:
 1. K->C: BootstrapRequestMessage
 2. C->K: ChallengeRequestMessage
 3. K->C: ChallengeResponseMessage
 4. K->K: BootstrapResponseMessage
-->

<wsdl:definitions                                                                    name="OctopusBootstrap"
targetNamespace="http://www.intertrust.com/services/OctopusBootstrap"
xmlns="http://schemas.xmlsoap.org/wsdl/"                    xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://www.intertrust.com/services/OctopusBootstrap"
xmlns:intf="http://www.intertrust.com/services/OctopusBootstrap"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tnstype="http://www.intertrust.com/services/OctopusBootstrap"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"      xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ob="http://www.intertrust.com/Octopus/Bootstrap/1.0"
xmlns:nc="http://www.intertrust.com/core">
 <wsdl:types>
  <schema                              targetNamespace="http://www.intertrust.com/services/OctopusBootstrap"
xmlns="http://www.w3.org/2001/XMLSchema">
   <!-- импорты -->
   <import                                    namespace="http://www.intertrust.com/Octopus/Bootstrap/1.0"
schemaLocation="./SimpleBootProtocol.xsd"/>
   <!-- элементы -->
   <element name="requestdata">
    <complexType>
     <!-- Это многоцикловый протокол без определения состояний (благодаря куки):
          клиент может послать BootstrapRequestMessage или
          ChallengeReponseMessage -->
     <choice>
      <element ref="ob:BootstrapRequestMessage"/>
      <element ref="ob:ChallengeResponseMessage"/>
     </choice>
    </complexType>
   </element>
   <element name="responsedata">
    <complexType>
     <!-- Это многоцикловый протокол без определения состояний (благодаря куки):
      сервер может в ответ послать ChallengeRequestMessage or
      BootstrapResponseMessage or an ErrorResponseMessage -->
     <choice>
      <element ref="ob:ChallengeRequestMessage"/>
      <element ref="ob:BootstrapResponseMessage"/>
      <element ref="ob:ErrorResponseMessage"/>
     </choice>
```

```
      </complexType>
    </element>
  </schema>
 </wsdl:типы>
 <!-- объявление сообщения -->
 <wsdl:message name="invokeRequest">
  <wsdl:part element="tnstype:requestdata" name="invokeRequest"/>
 </wsdl:message>
 <wsdl:message name="invokeResponse">
  <wsdl:part element="tnstype:responsedata" name="invokeResponse"/>
 </wsdl:message>
 <!-- объявление типа порта -->
 <wsdl:portType name="OctopusBootstrap">
  <wsdl:operation name="invoke">
   <wsdl:input message="impl:invokeRequest" name="invokeRequest"/>
   <wsdl:output message="impl:invokeResponse" name="invokeResponse"/>
  </wsdl:operation>
 </wsdl:portType>
 <!-- объявление привязки -->
 <wsdl:binding name="OctopusBootstrapSoapBinding" type="impl:OctopusBootstrap">
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="invoke">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="invokeRequest">
    <wsdlsoap:body    encodingStyle=""    namespace="http://www.intertrust.com/services/OctopusBootstrap"
use="literal"/>
   </wsdl:input>
   <wsdl:output name="invokeResponse">
    <wsdlsoap:body    encodingStyle=""    namespace="http://www.intertrust.com/services/OctopusBootstrap"
use="literal"/>
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
 <!-- объявление службы -->
 <wsdl:service name="OctopusBootstrapService">
  <wsdl:port binding="impl:OctopusBootstrapSoapBinding" name="OctopusBootstrap">
   <wsdlsoap:address location="http://localhost:8080/OctopusBootstrap/services/OctopusBootstrap"/>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Приложение D

Ниже представлен способ вычисления, не зависящий от кодирования, канонической последовательности байтов (CBS) для объектов, который используется, в предпочтительных вариантах осуществления, при вычислении дайджестов для использования цифрового подписывания объектов. Эта последовательность байтов не зависит от способа представления или передачи объектов, что позволяет использовать одни и те же значения дайджеста и подписи в системах, где используются множественные форматы кодирования (например, XML, ANS1), языки программирования, и т.п.

1. Алгоритм канонической последовательности байтов.

Алгоритм канонической последовательности байтов состоит в построении последовательностей байтов из значений полей. Каждое поле имеет значение простого типа или составного типа. Некоторые поля могут быть заданы как необязательные (поле может присутствовать или отсутствовать).

В одном варианте осуществления простые типы представляют собой целое число, строку, байт и логическое значение.

Составные типы состоят из одного или нескольких подполей, каждое из которых имеет значение простого или составного типа. Составные типы могут быть разнородными или однородными, в том смысле, что содержат одно или несколько значений подполей (простых или составных) разных типов (т.е., разнородные), или содержат одно или несколько значений подполей (простых или составных) одного типа (однородные).

Каноническая последовательность байтов поля получается путем применения правила кодирования к значению поля, когда поле всегда присутствует, или правила кодирования для необязательных полей, когда поле задано как необязательное. В нижеследующих описаниях правил кодирования, термин байт означает 8-битовое значение (октет).

1.1. Необязательные поля.

Если необязательное поле присутствует, его значение сериализуется как байтовое значение 1, после которого следует каноническая последовательность байтов значения поля. Если оно опущено, его значение сериализуется как байтовое значение 0.

1.2. Разнородный составной тип.

Каноническая последовательность байтов является сцеплением канонических последовательностей байтов значений всех подполей (необязательные поля не пропускаются, но сериализуются согласно правилу для необязательных полей).

1.3. Однородный составной тип.

Каноническая последовательность байтов представляет собой счетчик подполей, закодированный в виде последовательности из 4 байтов в обратном порядке следования байтов, после которого следует сцепление канонических последовательностей байтов значений всех подполей. Если счетчик подполей равен 0, то после 4-байтового счетчика полей ничего нет (в этом случае, все 4 байта имеют значение 0).

1.4. Целое число.

32-битовое целочисленное значение, закодированное в виде последовательности из 4 байтов, в обратном порядке следования байтов.

1.5. Строка.

Счетчик байтов	байт 0	байт 1	...

Строки представлены последовательностью байтов в кодировке UTF-8 (без символа конца строки). Каноническая последовательность байтов для строки состоит из (1) счетчика байтов строки, закодированного в виде последовательности из 4 байтов в обратном порядке следования байтов, после которого следует (2) последовательность байтов строки.

1.6. Байт.

8-битовое значение.

1.7. Логическое значение.

8-битовое значение: 0 означает ложь и 1 - истину.

2. Применение к объектам Octopus.

В одном варианте осуществления, каноническая последовательность байтов для объекта Octopus представляет собой сцепление канонических последовательностей байтов всех его полей, в том порядке, в каком они заданы в модели объекта.

Для разнородных составных типов, порядок полей указан в определении типа. Для однородных составных типов, порядок элементов указан в нижеследующих разделах.

Атрибуты

Поле "attributes" объекта рассматривается как безымянный атрибут типа "list" (это несортированный контейнер именованных атрибутов). Именованные атрибуты, содержащиеся в значении attributes типа "list", рассортированы лексикографически по их полю "name". Безымянные атрибуты, содержащиеся в значении attributes типа "array", не сортированы (они сериализуются в порядке их размещения в массиве).

Расширения

Внутренние расширения объекта рассортированы лексикографически по их полю 'id'. В одном варианте осуществления для внутренних расширений, поле 'extensionData' не используется при вычислении канонической последовательности байтов. Для таких расширений, если необходимо их включение в вычисление дайджеста с целью подписи, они будут содержать поле 'digest', представляющее дайджест фактических данных, переносимых в 'extensionData'. Для каждого типа данных расширения предусмотрено определение, которое позволяет вычислять его каноническую последовательность байтов.

Контроллер

Ссылки на ContentKey рассортированы лексикографически по их полю 'id' .

3. ScubaKeys.

Ключи в полях 'publicKeys', 'privateKeys' и 'secretKeys' рассортированы лексикографически по их полю 'id'.

4. Пример

```
Class X {
   int i;
   int j;
}

class A {
   int a[];
   string s;
}

class B extends A {
{X optional_x;}
X x;
(string toDiscardInCano;)
string s2;
}
```

Каноническая последовательность байтов экземпляра класса B, где a[] = {7,8,9}, s = "Abe", x =

{5,4}, s2="" и optioinal_x отсутствует, сериализуется следующим образом:

3	7	8	9	3	"Abc" как UTF-8	0	Cano(X)	0
4 байта	4 байта	4 байта	4 байта	4 байта	3 байта	1 байт	8 байтов	4 байта

где Cano(X) представляет собой:

5	4
4 байта	4 байта

Приложение E

Ниже приведен пример программы управления. В этом примере, лицензия указывает, что действие «воспроизведение» может быть разрешено, если состояние принадлежности (предусмотренное при регистрации) или состояние лицензии (предусмотренное при передаче лицензии) можно найти в базе данных состояний (именуемой в этом иллюстративном варианте осуществления базой данных "Seashell"). Лицензия также позволяет равноправному устройству запрашивать передачу лицензии. Эта передача разрешается, если два равноправных устройства находятся в данной степени близости друг от друга. Лицензия содержит агент, который задает состояние лицензии на равноправном устройстве.

В нижеследующих полях кода, "MovableDomainBoundLicense.asm" является главным объектом управления, "LicenseUtils/*" являются помощниками для лицензии, "GenericUtils/*" являются помощниками общего назначения, которые осуществляют, например, функции вычисления длины строки, сравнения строк, манипулирования стеком и/или т.п., и "ExtendedStatusBlockParameters/*" содержит XML-описание параметра расширенного блока состояний и соответствующее представление в виде ряда байтов, скомпилированных из XML.

E.1 MovableDomainBound.asm

```
;*************************************************************
;    Имя файла: MovableDomainBoundLicense.asm
;    Описание: Пример перемещаемой лицензии
;*************************************************************

; ========================
; константы
; ========================
.equ DEBUG_PRINT_SYSCALL,                1
.equ FIND_SYSCALL_BY_NAME_SYSCALL,       2
.equ SYSTEM_HOST_GET_OBJECT_SYSCALL,     3
.equ SYSTEM_HOST_SET_OBJECT_SYSCALL,     4
.equ SUCCESS,                            0
.equ FAILURE,                           -1
.equ ERROR_NO_SUCH_ITEM,                -6
.equ CONTAINER_IGNORED_ADDRESS,          1

; ========================
; включение
; ========================
.include "StrCmp.asm"
.include "PrintInt.asm"
.include "MembershipUtils.asm"
.include "LicenseStateUtils.asm"
```

```
;=========================
; данные
;=========================
.data

GetTrustedTimeFunctionName:
    .string "System.Host.GetTrustedTime"
GetTrustedTimeFunctionNumber:
    .long 0
ActionGrantedNoObligationXStatus:
    .long 0x00000000 ; глобальные флаги
    .long 0x00000000 ; категория = ACTION_GRANTED
    .long 0x00000000 ; подкатегория
    .long 0x00000000 ; локальные флаги
    .long 0x00000000 ; тип времени жизни кэша
    .long 0x00000000 ; значение времени жизни кэша
    .long 0x00000000 ; размер списка значений = 0
ActionDeniedXStatus:
    .long 0x00000000 ; глобальные флаги
    .long 0x00000001 ; категория = ACTION_DENIED
    .long 0x00000000 ; подкатегория
    .long 0x00000000 ; локальные флаги
    .long 0x00000000 ; тип времени жизни кэша
    .long 0x00000000 ; значение времени жизни кэша
    .long 0x00000000 ; размер списка значений = 0
TransferGrantedProximityNotChecked:
    .long 0x00000000 ; глобальные флаги
    .long 0x00000000 ; категория = ACTION_GRANTED
    .long 0x00000000 ; подкатегория
    .long 0x00000003 ; локальные флаги: Obligation и Callback Notices
    .long 0x00000000 ; тип времени жизни кэша
    .long 0x00000000 ; значение времени жизни кэша
    .include "TransferXStatusProximityCheckFailed.asm"
TransferGrantedProximityChecked:
    .long 0x00000000 ; глобальные флаги
    .long 0x00000000 ; категория = ACTION_GRANTED
    .long 0x00000000 ; подкатегория
    .long 0x00000003 ; локальные флаги: Obligation и Callback Notices
    .long 0x00000000 ; тип времени жизни кэша
    .long 0x00000000 ; значение времени жизни кэша
    .include "TransferXStatusProximityCheckSucceed.asm"
AgentContextPath:
    .string "Octopus/Agent/Session/ContextId"
AgentContextDesiredValue:
    .string "MoveStateContent0023"
AgentContextValue:
    .zeros 32
SinkProximityLastProbePath:
    .string "Octopus/Action/Parameters/Sink/Proximity/LastProbe"
SinkProximityLastProbeResult:
    .long -1
AgentProximityCheckedPath:
    .string "Octopus/Agent/Parameters/ProximityChecked"
AgentProximityCheckedValue:
    .long 0
ControllerTimestampAttributePath:
    .string "Octopus/Controller/Attributes/Import-time"
ControllerTimestampAttributeValue:
    .long 0

; отладка
.ifdef DEBUG
```

```
Controller.Timestamp.Query.Debug:
    .string "--------- Entering Controller.Timestamp.Query --------------\n"
Control.Actions.Play.Perform.Debug:
    .string "--------- Entering Control.Actions.Play.Perform --------------
\n"
Control.Agents.SetStateContent0023.Run.Debug:
    .string "--------- Entering Control.Agents.SetStateContent0023.Run -----
---------\n"
Control.Actions.Transfer.Perform.Debug:
    .string "--------- Entering Control.Actions.Transfer.Perform -----------
---\n"
Control.Agents.SetStateContent0023.OnAgentCompletion.Debug:
    .string                          "---------                          Entering
Control.Agents.SetStateContent0023.OnAgentCompletion --------------\n"
Transfer_OK_Proximity_Not_Checked.Debug:
    .string "####### Transfer_OK_Proximity_Not_Checked #######\n"
Transfer_OK_Proximity_Checked.Debug:
    .string "####### Transfer_OK_Proximity_Checked #######\n"
Agent_Failure.Debug:
    .string "####### Agent Failure #######\n"
Agent_Success.Debug:
    .string "####### Agent Success #######\n"
Action_Granted.Debug:
    .string "####### Action Granted #######\n"
Action_Denied.Debug:
    .string "####### Action Denied #######\n"

.endif

;=========================
; код
;=========================
.code

;
; Global.OnLoad
;
Global.OnLoad:
    ; получить functionNumber для GetTrustedTime
    PUSH @GetTrustedTimeFunctionName
    PUSH FIND_SYSCALL_BY_NAME_SYSCALL
    CALL
    DUP
    PUSH @GetTrustedTimeFunctionNumber
    POKE
    BRN OnLoad_Failed

    ; ok
    PUSH SUCCESS
    STOP

    ; неудача
OnLoad_Failed:
    PUSH FAILURE
    STOP

;
; Controller.Timestamp.Query
;
Controller.Timestamp.Query:
.ifdef DEBUG
    ;отладка
    PUSH @Controller.Timestamp.Query.Debug
    PUSH DEBUG_PRINT_SYSCALL
```

```
    CALL
.endif

    ; получить атрибут метки времени в объекте «контроллер»
    PUSH 4                                  ; ReturnBufferSize (4 байта)
    PUSH @ControllerTimestampAttributeValue ; ReturnBuffer (тип long)
    PUSH @ControllerTimestampAttributePath  ; Имя
    PUSH 0                                  ; Родитель = корневой контейнер
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    RET


;
; Control.Actions.Play.Check
;
Control.Actions.Play.Check:
;
; Control.Actions.Play.Perform
;
Control.Actions.Play.Perform:
    ; запросить пути к состоянию
    JSR MembershipStatePath.Query
    BRN Action_Denied

    JSR LicenseStatePath.Query
    BRN Action_Denied

.ifdef DEBUG
    ;отладка
    PUSH @Control.Actions.Play.Perform.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    ; если метка времени состояния принадлежности >
    ; метки времени контроллера
    JSR MembershipStateValue.Query
    BRN Action_Denied
    JSR Controller.Timestamp.Query
    BRN Action_Denied
    PUSH @MembershipStateValue
    PEEK
    PUSH @ControllerTimestampAttributeValue
    PEEK
    SUB
    BRP Action_Granted ; нам не нужно проверять состояние лицензии
                       ; в этом случае

    ; мы просто проверяем, присутствует ли состояние
    JSR LicenseStateValue.Query
    BRN Action_Denied

Action_Granted:
.ifdef DEBUG
    ;отладка
    PUSH @Action_Granted.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    PUSH @ActionGrantedNoObligationXStatus
    PUSH SUCCESS
    STOP
```

```
Action_Denied:
.ifdef DEBUG
    ;отладка
    PUSH @Action_Denied.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    PUSH @ActionDeniedXStatus
    PUSH SUCCESS
    STOP

;
; Control.Actions.Transfer.Check
;
Control.Actions.Transfer.Check:
;
; Control.Actions.Transfer.Perform
;
Control.Actions.Transfer.Perform:
    ; запросить пути к состоянию
    JSR MembershipStatePath.Query
    BRN Action_Denied

    JSR LicenseStatePath.Query
    BRN Action_Denied

.ifdef DEBUG
    ;отладка
    PUSH @Control.Actions.Transfer.Perform.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    ; получить близость, которая была проверена в последний раз
    PUSH 4                              ; ReturnBufferSize
    PUSH @SinkProximityLastProbeResult  ; ReturnBuffer
    PUSH @SinkProximityLastProbePath    ; Имя
    PUSH 0                              ; Родитель = корневой контейнер
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    ; если объект невидим, близость не была проверена.
    ; результирующий агент (см. файл TransferXStatusProximityCheckFailed.xml
    ; в строке 23). Агент сможет удостовериться, что агент является частью
    ; домена, до задания состояния
    BRN Transfer_OK_Proximity_Not_Checked

    ; проверить, что близость была проверена в последние 10 минут
    DROP                                ; мы знаем, что id типа есть long
    DROP                                ; мы знаем, что размер равен 4
    PUSH @GetTrustedTimeFunctionNumber
    PEEK
    CALL
    SWAP
    DROP                                ; нам просто нужно значение
    PUSH @SinkProximityLastProbeResult
    PEEK
    SUB
    PUSH 10
    SWAP
    SUB
    ; в последний раз близость была проверена более 10' назад: то же самое,
    ; как если бы она не проверялась вовсе (см. выше)
```

```
    BRN Transfer_OK_Proximity_Not_Checked

    ; близость проверена успешно
.ifdef DEBUG
    ;отладка
    PUSH @Transfer_OK_Proximity_Checked.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    PUSH @TransferGrantedProximityChecked
    PUSH SUCCESS
    STOP

Transfer_OK_Proximity_Not_Checked:
.ifdef DEBUG
    ; отладка
    PUSH @Transfer_OK_Proximity_Not_Checked.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    ; передать обратно RunAgentOnPeer Obligation (указывающий, что
    ; близость не была проверена) и OnAgentCompletion Callback
    PUSH @TransferGrantedProximityNotChecked
    PUSH SUCCESS
    STOP

;
; Control.Agents.SetStateContent0023.Run
;
Control.Agents.SetStateContent0023.Run:
    ; запросить пути к состоянию
    JSR MembershipStatePath.Query
    BRN Agent_Run_Failed

    JSR LicenseStatePath.Query
    BRN Agent_Run_Failed

.ifdef DEBUG
    ; отладка
    PUSH @Control.Agents.SetStateContent0023.Run.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    ; если равноправное устройство находится в домене,
    ; нам ничего не нужно делать
    JSR Membership.Check
    BRZ Agent_Success

    ; check that the context is set
    PUSH 32                             ; ReturnBufferSize
    PUSH @AgentContextValue             ; ReturnBuffer
    PUSH @AgentContextPath              ; Имя
    PUSH 0                              ; Родитель = корневой контейнер
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    ; check the result
    BRN Agent_Run_Failed
    DROP                                ; мы знаем, что id типа есть string
    DROP                                ; размер нам не важен
    PUSH @AgentContextValue
    PUSH @AgentContextDesiredValue
```

```
    JSR streq
    ; убедиться, что мы в хорошем контексте
    BRN Agent_Run_Failed

    ; проверить, успешно ли источник проверил приемник на близость
    PUSH 4                              ; ReturnBufferSize
    PUSH @AgentProximityCheckedValue    ; ReturnBuffer
    PUSH @AgentProximityCheckedPath     ; Имя
    PUSH 0                              ; Родитель = корневой контейнер
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    ; проверить результат
    BRN Agent_Run_Failed                ; этот параметр приложение всегда
                                        ; должно задавать при приеме агента

    DROP                                ; мы знаем, что id типа есть long
    DROP                                ; мы знаем, что размер равен 4
    PUSH @AgentProximityCheckedValue
    PEEK
    NOT
    BRZ Agent_Set_State

Agent_Run_Failed:
.ifdef DEBUG
    ;отладка
    PUSH @Agent_Failure.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    PUSH 0                              ; ReturnBlockSize
    PUSH 0                              ; ReturnBlockAddress
    PUSH FAILURE                        ; ResultCode
    STOP

Agent_Set_State:
    ; задать состояние
    JSR LicenseState.Set
    BRN Agent_Run_Failed

Agent_Success:
.ifdef DEBUG
    ;отладка
    PUSH @Agent_Success.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
.endif

    ; успех
    PUSH 0                              ; ReturnBlockSize
    PUSH 0                              ; ReturnBlockAddress
    PUSH SUCCESS                        ; ResultCode
    STOP

;
; обратный вызов Control.Agents.SetStateContent0023.OnAgentCompletion
; (типа RESET)
;
Control.Agents.SetStateContent0023.OnAgentCompletion:
.ifdef DEBUG
    ;отладка
    PUSH @Control.Agents.SetStateContent0023.OnAgentCompletion.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
```

```
.endif

    ; проверить, что код результата агента OK
    ; стек это:
    ; ... AgentResultCode CompletionStatusCode ArgumentsBlockSize Cookie
    DROP                                ; нам не нужны куки
    DROP                                ; нам не нужен размер блока
                                        ; аргументов
    BRN Action_Denied                   ; если выполнение агента не удалось,
                                        ; неудача
    BRN Action_Denied                   ; то же саме, как если бы агенту
                                        ; не удалось задать состояние на
                                        ; равноправном устройстве

    ; успех
    PUSH @ActionGrantedNoObligationXStatus
    PUSH SUCCESS
    STOP

Agent_Completion_Failed:
    PUSH FAILURE
    STOP

;========================
; экспорты
;========================
.export Global.OnLoad
.export Control.Actions.Play.Check
.export Control.Actions.Play.Perform
.export Control.Actions.Transfer.Check
.export Control.Actions.Transfer.Perform
.export Control.Agents.SetStateContent0023.Run
.export Control.Agents.SetStateContent0023.OnAgentCompletion
```

E.2 LicenseUtils

E.2.1 LicenseStateUtils.asm

```
;*************************************************************
;  Имя файла: LicenseStateUtils.asm
;  Описание: Утилиты для состояний лицензии
;*************************************************************;

;=========================
; данные
;=========================
.data
LicenseStatePathControlAttribute:
    .string "Octopus/Control/Attributes/LicenseStatePath"
LicenseStatePath:
    .zeros 256
LicenseStateValue:
    .long 0

; отладка
LicenseStatePath.Query.Debug:
  .string "--------- Entering LicenseStatePath.Query ----------
----\n"
LicenseStateValue.Query.Debug:
  .string "--------- Entering LicenseStateValue.Query ---------
-----\n"
LicenseState.Erase.Debug:
```

```
  .string "--------- Entering LicenseState.Erase --------------
\n"
LicenseState.Set.Debug:
  .string "--------- Entering LicenseState.Set --------------
\n"

;=========================
; код
;=========================
.code

;
; LicenseStatePath.Query
;
LicenseStatePath.Query:
    ;отладка
    PUSH @LicenseStatePath.Query.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    PUSH 256                                                    ;
ReturnBufferSize
    PUSH @LicenseStatePath                      ; ReturnBuffer
    PUSH @LicenseStatePathControlAttribute      ; Имя
    PUSH 0                                          ; Родитель =
корневой контейнер
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    RET

;
; LicenseStateValue.Query
;
LicenseStateValue.Query:
    ;отладка
    PUSH @LicenseStateValue.Query.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    PUSH 4                                  ; ReturnBufferSize (4
байта)
    PUSH @LicenseStateValue                   ; ReturnBuffer (тип
long)
    PUSH @LicenseStatePath                ; Имя
    PUSH 0                                  ; Родитель = корневой
контейнер
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL
    RET

;
; LicenseState.Erase
;
```

```
LicenseState.Erase:
    ;отладка
    PUSH @LicenseState.Erase.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    ; удалить локальное состояние
    PUSH 0                                          ; Размер объекта
(контейнер)
    PUSH 0                                             ; Тип объекта
(контейнер)
    PUSH 0                                       ; Удалить контейнер
    PUSH @LicenseStatePath                       ; Имя
    PUSH 0                                         ; Родитель = корневой
контейнер
    PUSH SYSTEM_HOST_SET_OBJECT_SYSCALL
    CALL
    RET

;
; LicenseState.Set
;
LicenseState.Set:
    ;отладка
    PUSH @LicenseState.Set.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    ; задать состояние
    PUSH 0                                          ; Размер объекта
(контейнер)
    PUSH 0                                             ; Тип объекта
(контейнер)
    PUSH CONTAINER_IGNORED_ADDRESS
    PUSH @LicenseStatePath                       ; Имя
    PUSH 0                                         ; Родитель = корневой
контейнер
    PUSH SYSTEM_HOST_SET_OBJECT_SYSCALL
    CALL
    RET
```

E.2.2 MembershipUtils.asm

```
;*********************************************************
;    Имя файла: MembershipUtils.asm
;     Описание: Утилиты для принадлежности к широковещательной
услуге
;
;*********************************************************;

;=========================
; данные
;=========================
.data
MembershipStatePathControlAttribute:
```

```
    .string "Octopus/Control/Attributes/MembershipStatePath"
MembershipStatePath:
    .zeros 256
MembershipStateValue:
    .long 0

; отладка
intStrOutput:
    .string "............"
MembershipStatePath.Query.Debug:
  .string "--------- Entering MembershipStatePath.Query -------
----\n"
MembershipStateValue.Query.Debug:
   .string "--------- Entering MembershipStateValue.Query -----
----\n"
Membership.Check.Debug:
    .string "--------- Entering Membership.Check -------------
\n"
Membership_Check_Success.Debug:
    .string "####### Membership Check Success #######\n"
Membership_Check_Failure.Debug:
    .string "####### Membership Check Failure #######\n"
MembershipPath.Debug:
    .string "MembershipState path: "
MembershipGetObjOutput.Debug:
    .string "MembershipState get объект возвращает: "
Membership_Expired.Debug:
    .string "MembershipState has expired. Check the Value of
the Membership SeaShell token against the местное время."
NewlineString:
    .string "\n"

;=========================
; код
;=========================
.code

;
; MembershipStatePath.Query
;
MembershipStatePath.Query:
    ;отладка
    PUSH @MembershipStatePath.Query.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    PUSH 256                                                    ;
ReturnBufferSize
    PUSH @MembershipStatePath                  ; ReturnBuffer
    PUSH @MembershipStatePathControlAttribute ; Имя
    PUSH 0                                          ; Родитель =
корневой контейнер
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
```

```
    CALL
    RET

;
; MembershipStateValue.Query
;
MembershipStateValue.Query:
    ;отладка
    PUSH @MembershipStateValue.Query.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
    PUSH @MembershipPath.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
    PUSH @MembershipStatePath
    PUSH DEBUG_PRINT_SYSCALL
    CALL
    PUSH @NewlineString
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    PUSH 4                                  ; ReturnBufferSize (4
байта)
    PUSH @MembershipStateValue              ; ReturnBuffer (тип
long)
    PUSH @MembershipStatePath           ; Имя
    PUSH 0                                  ; Родитель = корневой
контейнер
    PUSH SYSTEM_HOST_GET_OBJECT_SYSCALL
    CALL

    PUSH @MembershipGetObjOutput.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
    ; печатать результат - сначала преобразовать int в string
    DUP
    PUSH @intStrOutput
    ADD
    SWAP
    JSR printInt
    ; вызвать печать результата
    PUSH @intStrOutput
    PUSH DEBUG_PRINT_SYSCALL
    CALL
    PUSH @NewlineString
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    RET

;
; Membership.Check
;
```

```
Membership.Check:
    ;отладка
    PUSH @Membership.Check.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    ; запросить состояние принадлежности
    JSR MembershipStateValue.Query
    ; проверить успешность
    BRN Membership_Check_Failed

    ;  проверить,  что  время  <  извлеченного  в  состоянии
принадлежности
    PUSH @MembershipStateValue ; метка времени
    PEEK
    PUSH @GetTrustedTimeFunctionNumber
    PEEK
    CALL
    SWAP
    DROP ; нам просто нужно значение (не оценка)
    SUB
    BRN Membership_Expired

    ; успех
    ;отладка
    PUSH @Membership_Check_Success.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    PUSH SUCCESS
    RET

Membership_Expired:
    ;отладка
    PUSH @Membership_Expired.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL
    BRA Membership_Check_Failed

Membership_Check_Failed:
    ;отладка
    PUSH @Membership_Check_Failure.Debug
    PUSH DEBUG_PRINT_SYSCALL
    CALL

    PUSH FAILURE
    RET
```

E.3 GlobalUtils

E.3.1 IntUtils.asm

```
;*********************************************************
;    Имя файла: IntUtils.asm
;    Описание: утилиты для сравнения 2 целых чисел
;*********************************************************
```

```
;=======================
; включение
;=======================
.include "StackUtils.asm"

;=======================
; код
;=======================
.code

;
; min
;
; вычисляет мимнимум между 2 целыми числами
;
; вход: ... a b
; выход:  ... a<b?a:b
;
min:
    DUP             ; ... a b b
    PUSH 2
    JSR pick        ; ... a b b a
    CMP             ; ... a b cmp_result
    BRN Swap_Stack  ; ... a b
    DROP
    RET             ; a

Swap_Stack:
    SWAP
    DROP
    RET             ; b

;
; max
;
; вычисляет максимум между 2 целыми числами
;
; вход: ... a b
; выход:  ... a>b?a:b
;
max:
    DUP             ; ... a b b
    PUSH 2
    JSR pick        ; ... a b b a
    CMP             ; ... a b cmp_result
    NEG
    BRN Swap_Stack
    DROP
    RET             ; a
```

E.3.2 PrintInt.asm

```
;*************************************************************
;    Имя файла: PrintInt.asm
;    Описание: преобразует целое число (знаковое или беззнаковое) в строку
;*************************************************************

;; Примечание: необходимо, чтобы "StackUtils.asm" уже был включен

; выбор данных
      .data

; выбор кода
      .code

; преобразует целое число в представление строки
```

```
; параметры: dest, int
printInt:
      ; дублировать параметр dest
      SWAP
      DUP
      PUSH 2
      JSR pick
      ; STACK: origval, startstring, startstring, unsignedval
      ; теперь у нас есть дополнительная копия целого числа на
      ; дне, которую мы будем использовать для проверки
      ; исходного знака
printIntLoop:
      ; получить одну цифру
      DUP
      PUSH 10
      MOD
      ; преобразовать в ascii
      PUSH 48 ; ASCII для '0'
      ADD
      ; получить адрес для выходного буфера
      PUSH 2
      JSR pick
      POKEB ; печатать в буфер
      ; перемещение указателя нашего буфера
      SWAP
      PUSH 1
      ADD
      SWAP
      ; разделить на 10 и посмотреть, где мы
      PUSH 10
      DIV
      DUP
      BRP printIntLoop
      DROP ; избавиться от 0 на вершине
      ; STACK = orignum, startofstring, endofstring
      ; если исходное число отрицательно, поставить знак минус
      ; завершить символом конца строки
      DUP
      PUSH 0
      SWAP
      POKEB
      ; переместить конец строки на 1 вверх,
      ; чтобы не перебрасывать символ конца строки
      PUSH 1
      SUB
      ; готово: нужно просто перевернуть строку
fliploop:
      ; получить второй байт
      DUP
      PEEKB
      ; получить первый байт
      PUSH 2
      JSR pick
      PEEKB
      ; поставить первый байт на место последнего
      PUSH 2
      JSR pick
      POKEB
      ; поставить последний байт на место первого
      PUSH 2
      JSR pick
      POKEB
      ; переместить указатель конца на единицу вверх
      PUSH 1
```

```
      SUB
      ; переместить указатель начала на единицу вниз
      SWAP
      PUSH 1
      ADD
      SWAP
      ; посмотреть, встретились ли указатели
      ; сначала нужно дублировать значения
      DUP
      PUSH 2
      JSR pick
      SUB
      BRP fliploop
      ; избавиться от некоторых наносов в стеке
      DROP
      DROP
      DROP
      RET
```

E.3.3 StackUtils.asm

```
;*************************************************************
;     Имя файла: StackUtils.asm
;     Описание: функции утилит стека, пришедшие из FORTH
;
;*************************************************************

.ifndef _STACK_UTILS_
.define _STACK_UTILS_

;=========================
; код
;=========================
.code

;
; over
;
; копировать второй элемент стека
;
; вход: ... a b
; выход:  ... a b a
;
over:
      PUSH 1
      JSR pick
      RET

;
; pick
;
; вход: ... v3 v2 v1 v0 N
; выход: ... v3 v2 v1 v0 vN
;
pick:
      PUSH 1
      ADD
      PUSH 4
```

```
      MUL
      PUSHSP
      ADD
      PEEK
      RET

.endif ; _STACK_UTILS_
```

E.3.4 StdLib.asm

```
;  Стандартная библиотека для Plankton
      .equ HEAP_ADDR, 16

; выбор дданных
; выбор кода
      .code
strlen:
      DUP
loop:
      DUP
      PEEKB
      BRZ done
      PUSH 1
      ADD
      BRA loop

done:
      SWAP
      SUB
      RET

      .export strlen
```

E.3.5 StrCmp.asm

```
;*************************************************************
;    Имя файла: StrCmp.asm
;    Описание: тесты streq на равенство двух строк
;*************************************************************

.ifndef _STR_CMP_
.define _STR_CMP_

;=========================
; включение
;=========================
.include "StackUtils.asm"

;=========================
; код
;=========================
.code

;
; streq
;
; тест на равенство двух строк
;
; вход: ... @str1 @str2
; выход: ... res (res = 0 if strings are equal, -1 otherwise)
;
streq:
```

```
      ; получить смещение между 2 строками
      JSR over
      SUB
      SWAP            ; ... смещение @str1

streqloop:
      ; получить текущий символ str1
      DUP
      PEEKB           ; ... смещение @str1 char1

      ; получить текущий символ str2
      JSR over        ; ... смещение @str1 char1 @str1
      PUSH 3
      JSR pick        ; ... смещение @str1 char1 @str1 offset
      ADD
      PEEKB           ; ... смещение @str1 char1 char2

      ; теперь сравнить два символа
      JSR over
      SUB             ; ... смещение @str1 char1 char1-char2
      ; fail if the char1 != char2
      NOT
      BRZ streqfailure
      ; если char1 равен 0 (char1 == char2 == 0), готово
      BRZ streqsuccess        ; ... смещение @str1

      ; увеличить указатель @str1 и вернуться к началу цикла
      PUSH 1
      ADD
      BRA streqloop

streqfailure:
      ; ... смещение @str1 char1
      DROP
      DROP
      DROP
      PUSH -1
      RET

streqsuccess:
      ; ... смещение @str1
      DROP
      DROP
      PUSH 0
      RET

.endif ; _STR_CMP_
```

E.4 ExtendedStatusBlock Parameters

E.4.1 TransferXStatusProximityCheckSucceeded.xml

```
<ValueListBlock>
 <ValueBlock type="Parameter">
  <ParameterBlock name="Obligations">
   <ValueBlock type="ValueList">
    <ValueListBlock>
     <ValueBlock type="ExtendedParameter">
      <ParameterBlock name="RunAgentOnPeer" flags="1">
       <ValueBlock type="ValueList">
```

```
<ValueListBlock>
    <!-- ID объекта управления -->
    <ValueBlock type="String">urn:marlin:control:0023</ValueBlock>
    <!-- Имя агента -->
    <ValueBlock type="String">SetStateContent0023</ValueBlock>
    <!-- id экземпляра -->
    <ValueBlock type="Integer">240343</ValueBlock>
    <!-- ID контекста -->
    <ValueBlock type="String">MoveStateContent0023</ValueBlock>
    <!-- дополнительные параметры -->
   <ValueBlock type="ValueList">
    <ValueListBlock>
     <ValueBlock type="Parameter">
          <!-- Единственное, что меняется с
          TransferXStatusProximityCheckFailed.xml -->
       <ParameterBlock name="ProximityChecked">
        <ValueBlock type="Integer">1</ValueBlock>
       </ParameterBlock>
     </ValueBlock>
    </ValueListBlock>
   </ValueBlock>
  </ValueListBlock>
 </ValueBlock>
</ParameterBlock>
</ValueBlock>
<ValueBlock type="Parameter">
 <ParameterBlock name="Callbacks">
  <ValueBlock type="ValueList">
   <ValueListBlock>
    <ValueBlock type="ExtendedParameter">
     <ParameterBlock name="OnAgentCompletion" flags="1">
      <ValueBlock type="ValueList">
       <ValueListBlock>
          <!-- id экземпляра агента -->
          <ValueBlock type="String">240343</ValueBlock>
          <!-- Процедура обратного вызова -->
        <ValueBlock type="ValueList">
         <ValueListBlock>
```

```
                    <!-- Сброс  -->
                    <ValueBlock type="Integer">0</ValueBlock>
                    <!-- Имя  -->
                    <ValueBlock
type="String">Control.Agents.SetStateContent0023.OnAgentCompletion</ValueBlock>
                    <!-- куки  -->
                    <ValueBlock type="Integer">0</ValueBlock>
                </ValueListBlock>
              </ValueBlock>
            </ValueListBlock>
          </ValueBlock>
        </ParameterBlock>
      </ValueBlock>
    </ValueListBlock>
  </ValueBlock>
 </ParameterBlock>
</ValueBlock>
</ValueListBlock>
```

E.4.2 TransferXStatusProximityCheckFailed.xml

```
<ValueListBlock>
 <ValueBlock type="Parameter">
  <ParameterBlock name="Obligations">
   <ValueBlock type="ValueList">
    <ValueListBlock>
     <ValueBlock type="ExtendedParameter">
      <ParameterBlock name="RunAgentOnPeer" flags="1">
       <ValueBlock type="ValueList">
        <ValueListBlock>
                    <!-- ID объекта управления -->
                    <ValueBlock type="String">urn:marlin:control:0023</ValueBlock>
                    <!-- имя агента -->
                    <ValueBlock type="String">SetStateContent0023</ValueBlock>
                    <!-- id экземпляра  -->
                    <ValueBlock type="Integer">240343</ValueBlock>
                    <!-- ID контекста  -->
                    <ValueBlock type="String">MoveStateContent0023</ValueBlock>
                    <!-- дополнительные параметры  -->
         <ValueBlock type="ValueList">
          <ValueListBlock>
           <ValueBlock type="Parameter">
                <!-- Единственое, что изменяется с TransferXStatusProximityCheckSucceed.xml -->
            <ParameterBlock name="ProximityChecked">
```

```
                        <ValueBlock type="Integer">0</ValueBlock>
                    </ParameterBlock>
                  </ValueBlock>
                 </ValueListBlock>
               </ValueBlock>
              </ValueListBlock>
             </ValueBlock>
            </ParameterBlock>
           </ValueBlock>
          </ValueListBlock>
         </ValueBlock>
        </ParameterBlock>
       </ValueBlock>
       <ValueBlock type="Parameter">
        <ParameterBlock name="Callbacks">
         <ValueBlock type="ValueList">
          <ValueListBlock>
           <ValueBlock type="ExtendedParameter">
            <ParameterBlock name="OnAgentCompletion" flags="1">
             <ValueBlock type="ValueList">
              <ValueListBlock>
                    <!-- id экземпляра агента  -->
                    <ValueBlock type="String">240343</ValueBlock>
                    <!-- Процедура обратного вызова  -->
                <ValueBlock type="ValueList">
                 <ValueListBlock>
                    <!-- Сброс  -->
                    <ValueBlock type="Integer">0</ValueBlock>
                    <!-- Имя  -->
                    <ValueBlock
type="String">Control.Agents.SetStateContent0023.OnAgentCompletion</ValueBlock>
                    <!-- куки  -->
                    <ValueBlock type="Integer">0</ValueBlock>
                 </ValueListBlock>
                </ValueBlock>
               </ValueListBlock>
              </ValueBlock>
             </ParameterBlock>
            </ValueBlock>
           </ValueListBlock>
          </ValueBlock>
         </ParameterBlock>
```

</ValueBlock>
</ValueListBlock>

E.4.3 TransferXStatusProximityCheckSucceeded.asm

```
0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x20
0x00 0x00 0x00 0x0C 0x4F 0x62 0x6C 0x69 0x67 0x61 0x74 0x69 0x6F
0x6E 0x73 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x0C 0x00 0x00
0x00 0x01 0x00 0x00 0x00 0x05 0x00 0x00 0x00 0x04 0x00 0x00 0x00
0x01 0x00 0x00 0x00 0x0F 0x52 0x75 0x6E 0x41 0x67 0x65 0x6E 0x74
0x4F 0x6E 0x50 0x65 0x65 0x72 0x00 0x00 0x00 0x00 0x07 0x00 0x00
0x00 0x92 0x00 0x00 0x00 0x05 0x00 0x00 0x00 0x02 0x00 0x00 0x00
0x18 0x75 0x72 0x6E 0x3A 0x6D 0x61 0x72 0x6C 0x69 0x6E 0x3A 0x63
0x6F 0x6E 0x74 0x72 0x6F 0x6C 0x3A 0x30 0x30 0x32 0x33 0x00 0x00
0x00 0x00 0x02 0x00 0x00 0x00 0x14 0x53 0x65 0x74 0x53 0x74 0x61
0x74 0x65 0x43 0x6F 0x6E 0x74 0x65 0x6E 0x74 0x30 0x30 0x32 0x33
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x03 0xAA 0xD7
0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x15 0x4D 0x6F 0x76 0x65 0x53
0x74 0x61 0x74 0x65 0x43 0x6F 0x6E 0x74 0x65 0x6E 0x74 0x30 0x30
0x32 0x33 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x25 0x00 0x00
0x00 0x01 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x1D 0x00 0x00 0x00
0x11 0x50 0x72 0x6F 0x78 0x69 0x6D 0x69 0x74 0x79 0x43 0x68 0x65
0x63 0x6B 0x65 0x64 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04
0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x1E 0x00
0x00 0x00 0x0A 0x43 0x61 0x6C 0x6C 0x62 0x61 0x63 0x6B 0x73 0x00
0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x0C 0x00 0x00 0x00 0x01 0x00
0x00 0x00 0x05 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x01 0x00 0x00
0x00 0x12 0x4F 0x6E 0x41 0x67 0x65 0x6E 0x74 0x43 0x6F 0x6D 0x70
0x6C 0x65 0x74 0x69 0x6F 0x6E 0x00 0x00 0x00 0x00 0x07 0x00 0x00
0x00 0x6C 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x02 0x00 0x00 0x00
0x07 0x32 0x34 0x30 0x33 0x34 0x33 0x00 0x00 0x00 0x00 0x07 0x00
0x00 0x00 0x55 0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00
0x35 0x43 0x6F 0x6E 0x74 0x72 0x6F 0x6C 0x2E 0x41 0x67 0x65 0x6E
0x74 0x73 0x2E 0x53 0x65 0x74 0x53 0x74 0x61 0x74 0x65 0x43 0x6F
0x6E 0x74 0x65 0x6E 0x74 0x30 0x30 0x32 0x33 0x2E 0x4F 0x6E 0x41
0x67 0x65 0x6E 0x74 0x43 0x6F 0x6D 0x70 0x6C 0x65 0x74 0x69 0x6F
0x6E 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x00 0x00
0x00
```

E.4.4 TransferXStatusProximityCheckFailed.asm

```
0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x20
0x00 0x00 0x00 0x0C 0x4F 0x62 0x6C 0x69 0x67 0x61 0x74 0x69 0x6F
0x6E 0x73 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x0C 0x00 0x00
0x00 0x01 0x00 0x00 0x00 0x05 0x00 0x00 0x00 0x04 0x00 0x00 0x00
```

```
0x01 0x00 0x00 0x00 0x0F 0x52 0x75 0x6E 0x41 0x67 0x65 0x6E 0x74
0x4F 0x6E 0x50 0x65 0x65 0x72 0x00 0x00 0x00 0x00 0x07 0x00 0x00
0x00 0x92 0x00 0x00 0x00 0x05 0x00 0x00 0x00 0x02 0x00 0x00 0x00
0x18 0x75 0x72 0x6E 0x3A 0x6D 0x61 0x72 0x6C 0x69 0x6E 0x3A 0x63
0x6F 0x6E 0x74 0x72 0x6F 0x6C 0x3A 0x30 0x30 0x32 0x33 0x00 0x00
0x00 0x00 0x02 0x00 0x00 0x00 0x14 0x53 0x65 0x74 0x53 0x74 0x61
0x74 0x65 0x43 0x6F 0x6E 0x74 0x65 0x6E 0x74 0x30 0x30 0x32 0x33
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x03 0xAA 0xD7
0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x15 0x4D 0x6F 0x76 0x65 0x53
0x74 0x61 0x74 0x65 0x43 0x6F 0x6E 0x74 0x65 0x6E 0x74 0x30 0x30
0x32 0x33 0x00 0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x25 0x00 0x00
0x00 0x01 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x1D 0x00 0x00 0x00
0x11 0x50 0x72 0x6F 0x78 0x69 0x6D 0x69 0x74 0x79 0x43 0x68 0x65
0x63 0x6B 0x65 0x64 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x1E 0x00
0x00 0x00 0x0A 0x43 0x61 0x6C 0x6C 0x62 0x61 0x63 0x6B 0x73 0x00
0x00 0x00 0x00 0x07 0x00 0x00 0x00 0x0C 0x00 0x00 0x00 0x01 0x00
0x00 0x00 0x05 0x00 0x00 0x00 0x04 0x00 0x00 0x00 0x01 0x00 0x00
0x00 0x12 0x4F 0x6E 0x41 0x67 0x65 0x6E 0x74 0x43 0x6F 0x6D 0x70
0x6C 0x65 0x74 0x69 0x6F 0x6E 0x00 0x00 0x00 0x00 0x07 0x00 0x00
0x00 0x6C 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x02 0x00 0x00 0x00
0x07 0x32 0x34 0x30 0x33 0x34 0x33 0x00 0x00 0x00 0x00 0x07 0x00
0x00 0x00 0x55 0x00 0x00 0x00 0x03 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x00 0x00 0x00
0x35 0x43 0x6F 0x6E 0x74 0x72 0x6F 0x6C 0x2E 0x41 0x67 0x65 0x6E
0x74 0x73 0x2E 0x53 0x65 0x74 0x53 0x74 0x61 0x74 0x65 0x43 0x6F
0x6E 0x74 0x65 0x6E 0x74 0x30 0x30 0x32 0x33 0x2E 0x4F 0x6E 0x41
0x67 0x65 0x6E 0x74 0x43 0x6F 0x6D 0x70 0x6C 0x65 0x74 0x69 0x6F
0x6E 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x00 0x00
0x00
```

Хотя вышеприведенное подробное описание было предоставлено для пояснения изобретения, очевидно, что возможны определенные изменения и модификации в пределах объема прилагаемой формулы изобретения. Заметим, что существует много альтернативных способов реализации описанных здесь процессов и устройств. Соответственно, настоящие варианты осуществления следует рассматривать как иллюстративные и неограничительные, и изобретение не подлежит ограничению рассмотренными здесь деталями, но допускает модификации в рамках объема и эквивалентов прилагаемой формулы изобретения.

ФОРМУЛА ИЗОБРЕТЕНИЯ

1. Способ авторизации доступа к фрагменту электронного контента на компьютерной системе хоста, при этом способ содержит этапы, на которых

принимают запрос от пользователя компьютерной системы хоста для доступа к фрагменту электронного контента,

извлекают лицензию, связанную с фрагментом электронного контента, причем лицензия содержит объект управления, объект контроллера, объект протектора и объект ключа контента,

извлекают первую программу управления из объекта управления и

выполняют первую программу управления с использованием механизма управления цифровыми правами, выполняющегося на компьютерной системе хоста, для определения, можно ли удовлетворить запрос, причем при выполнении программы управления оценивают один или несколько объектов связи, причем каждый объект связи представляет соотношение между двумя сущностями, причем по меньшей мере один из одного или нескольких объектов связи содержит вторую программу управления, и при оценивании одного или нескольких объектов связи выполняют вторую программу управления с использованием механизма управления цифровыми правами для определения, действительна ли связь, причем при выполнении определяют, выполняются ли одно или несколько условий, выраженных программой управления.

2. Способ по п.1, в котором объект контроллера способен безопасно связывать объект управления с объектом ключа контента.

3. Способ по п.1, в котором объект протектора способен безопасно связывать объект ключа контен-

та с фрагментом электронного контента.

4. Способ по п.1, в котором по меньшей мере одно из одного или нескольких условий содержит требование, чтобы текущее время было до заранее заданного времени.

5. Способ по п.1, в котором по меньшей мере одно из одного или нескольких условий содержит требование, чтобы текущее время было после определенного времени.

6. Способ по п.1, в котором по меньшей мере одно из одного или нескольких условий содержит требование, чтобы вторая программа управления предварительно не была выполнена больше заранее заданного числа раз.

7. Способ по п.1, в котором по меньшей мере одно из одного или нескольких условий содержит требование, чтобы счетчик, хранящийся в памяти, не превышал заранее заданного значения.

8. Способ по п.1, в котором по меньшей мере одно из одного или нескольких условий содержит требование, чтобы заранее заданное событие ранее не происходило.

9. Способ по п.1, в котором по меньшей мере одно из одного или нескольких условий содержит требование, чтобы компьютерная система хоста имела одну или несколько заранее заданных характеристик.

10. Способ по п.1, в котором по меньшей мере одно из одного или нескольких условий содержит требование, чтобы программное обеспечение, выполняющееся на компьютерной системе хоста для представления фрагмент электронного контента, было неспособно экспортировать фрагмент электронного контента на заранее заданный интерфейс.

11. Способ авторизации выполнения данного действия на фрагменте электронного контента, при этом способ содержит этапы, на которых

выполняют первую программу управления с использованием виртуальной машины, выполняющейся на первом механизме управления цифровыми правами, причем первая программа управления способна определять, может ли данное действие осуществляться на фрагменте электронного контента, причем первая программа управления способна оценивать первое множество из одного или нескольких условий, которые должны выполняться для того, чтобы осуществление данного действия было авторизовано, причем по меньшей мере одно из первого множества из одного или нескольких условий содержит требование, чтобы один или несколько объектов связи были доступны механизму управления цифровыми правами, причем объекты связи логически связывают первый узел, представляющий первую сущность, со вторым узлом, представляющим вторую сущность,

извлекают один или несколько объектов связи, причем каждый из объектов связи выражает соотношение между двумя сущностями, и по меньшей мере один из объектов связи включает в себя вторую программу управления, причем вторая программа управления способна оценивать второе множество из одного или нескольких условий, которые должны выполняться, чтобы по меньшей мере один объект связи можно было считать действительным, и

используют механизм управления цифровыми правами для выполнения второй программы управления.

12. Способ по п.11, в котором первое множество из одного или нескольких условий включает в себя условие, относящееся к времени.

13. Способ по п.11, в котором второе множество из одного или нескольких условий включает в себя условие, относящееся к времени.

14. Способ по п.11, в котором по меньшей мере одно из первого множества условий и второго множества условий содержит требование, чтобы счетчик, хранящийся в памяти, не превышал заранее заданного значения.

15. Способ авторизации доступа к фрагменту электронного контента в системе управления цифровыми правами, содержащей механизм управления цифровыми правами, включающий в себя виртуальную машину, причем способ содержит этапы, на которых

принимают запрос на доступ к фрагменту электронного контента или иное его использование,

идентифицируют лицензию, связанную с фрагментом электронного контента, причем лицензия содержит программу управления и ключ контента,

выполняют программу управления с использованием виртуальной машины,

получают выход виртуальной машины, причем выход указывает, что запрашиваемый доступ или иное использование фрагмента электронного контента авторизован(о), пока выполняется обязательство,

определяют, что приложение хоста способно выполнять обязательство, и

разрешают запрашиваемый доступ или иное использование фрагмента электронного контента при выполнении обязательства, включающего в себя использование ключа контента для дешифрования фрагмента электронного контента.

16. Способ по п.15, в котором обязательство содержит представление фрагмента электронного контента в формате пониженного качества.

17. Способ по п.15, в котором обязательство содержит запись информации аудита, относящейся к доступу или иному использованию фрагмента электронного контента, и передачу информации аудита в удаленное место.

18. Способ по п.15, в котором обязательство содержит требование, чтобы указанные функции среды хоста, в которой выполняется система управления цифровыми правами, были отключены.

19. Способ по п.18, в котором фрагмент электронного контента включает в себя рекламу и в котором указанные функции включают в себя возможность перемотки вперед и назад в ходе представления фрагмента электронного контента.

20. Способ по п.18, в котором указанные функции включают в себя возможность экспорта фрагмента электронного контента в определенную технологию.

21. Способ авторизации доступа к фрагменту электронного контента в системе хоста, содержащей механизм управления цифровыми правами, включающий в себя виртуальную машину, при этом способ содержит этапы, на которых

принимают запрос на доступ к фрагменту электронного контента или иное его использование,

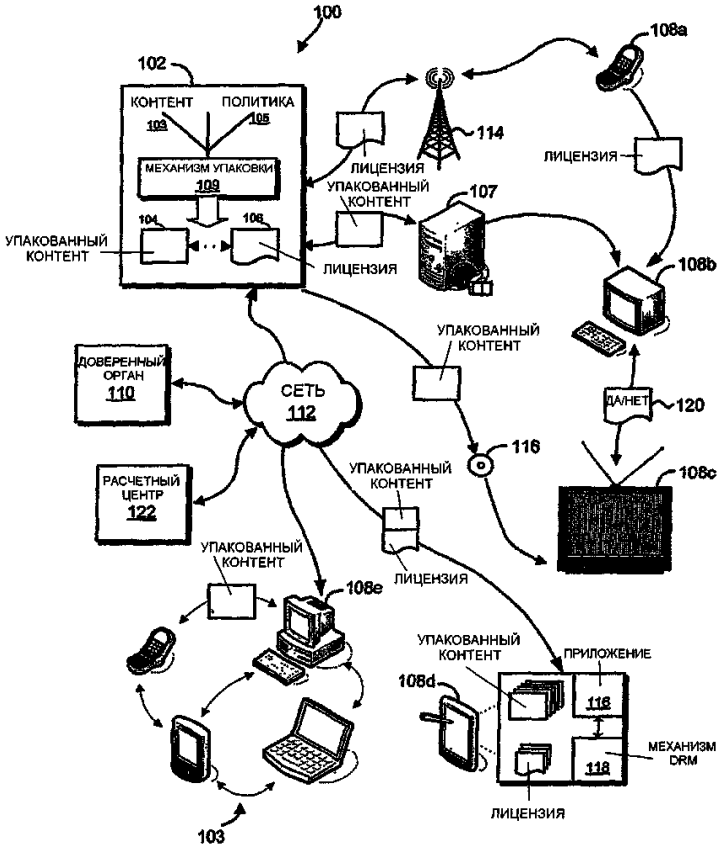
идентифицируют лицензию, связанную с фрагментом электронного контента, причем лицензия содержит программу управления и ключ контента,

выполняют программу управления с использованием виртуальной машины,

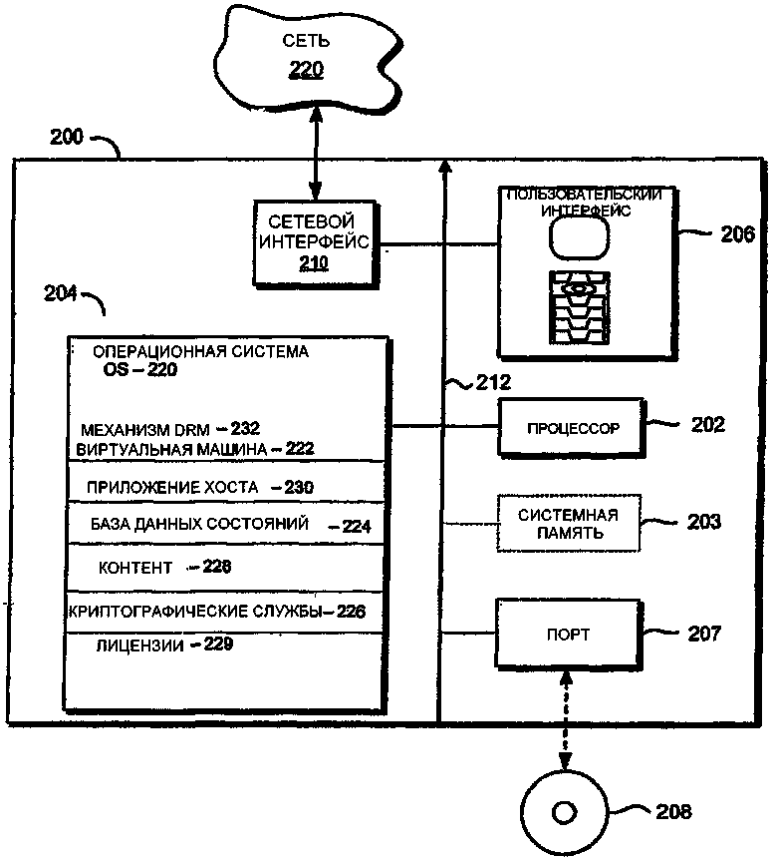
определяют, что запрашиваемый доступ или иное использование фрагмента электронного контента можно авторизовать, пока выполняется обязательство,

определяют, что система хоста не способна выполнять обязательство, и

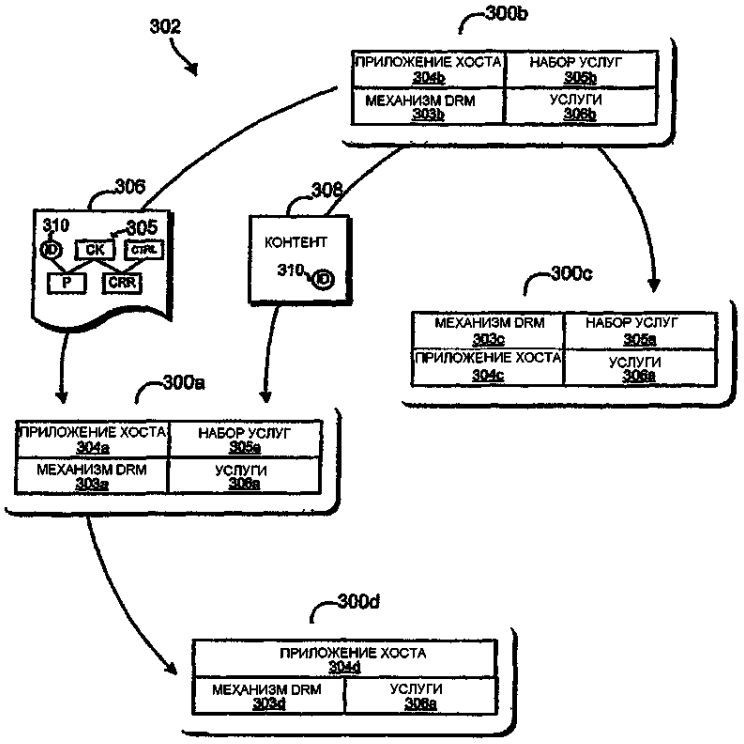
отказывают в запрашиваемом доступе или ином использовании фрагмента электронного контента.



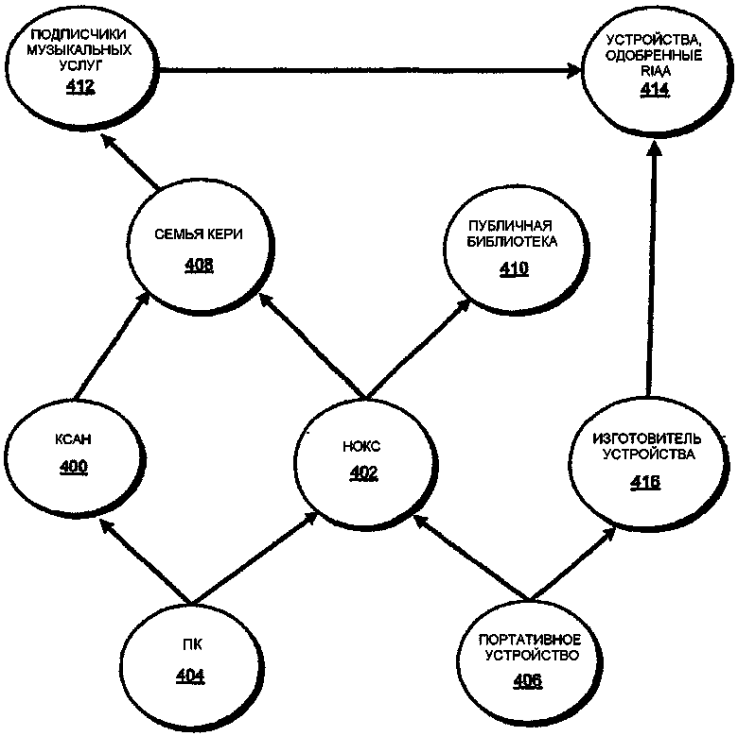
Фиг. 1


СЕТЬ
220
200
СЕТЕВОЙ ИНТЕРФЕЙС
210
ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС
206
204
ОПЕРАЦИОННАЯ СИСТЕМА
OS – 220
МЕХАНИЗМ DRM – 232
ВИРТУАЛЬНАЯ МАШИНА – 222
ПРИЛОЖЕНИЕ ХОСТА – 230
БАЗА ДАННЫХ СОСТОЯНИЙ – 224
КОНТЕНТ – 228
КРИПТОГРАФИЧЕСКИЕ СЛУЖБЫ – 226
ЛИЦЕНЗИИ – 229
212
ПРОЦЕССОР
202
СИСТЕМНАЯ ПАМЯТЬ
203
ПОРТ
207
208

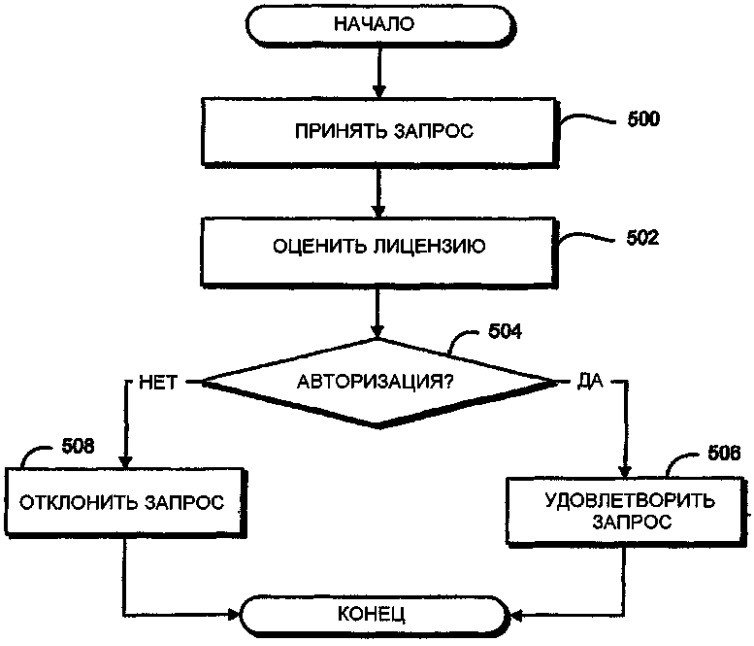
Фиг. 2


302
300b
ПРИЛОЖЕНИЕ ХОСТА 304b
НАБОР УСЛУГ 305b
МЕХАНИЗМ DRM 303b
УСЛУГИ 306b
306
310
305
ID
CK
CTRL
P
CRR
308
КОНТЕНТ
310
300c
МЕХАНИЗМ DRM 303c
НАБОР УСЛУГ 305a
ПРИЛОЖЕНИЕ ХОСТА 304c
УСЛУГИ 306a
300a
ПРИЛОЖЕНИЕ ХОСТА 304a
НАБОР УСЛУГ 305a
МЕХАНИЗМ DRM 303a
УСЛУГИ 306a
300d
ПРИЛОЖЕНИЕ ХОСТА 304d
МЕХАНИЗМ DRM 303d
УСЛУГИ 306a

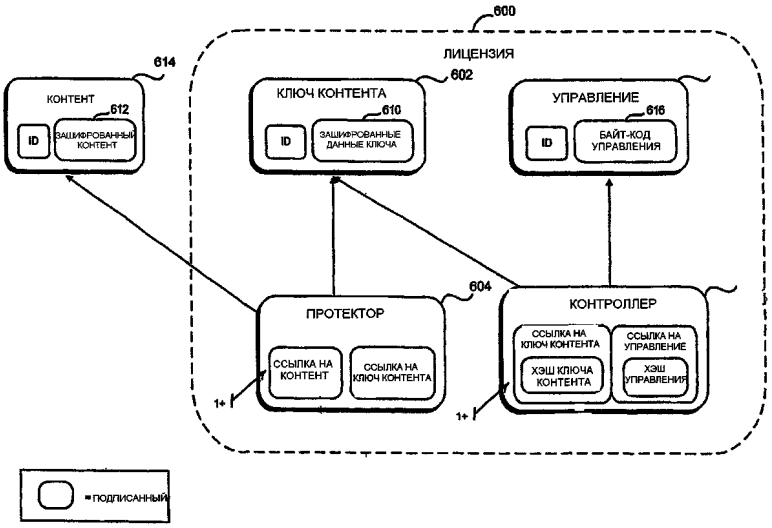
Фиг. 3


ПОДПИСЧИКИ МУЗЫКАЛЬНЫХ УСЛУГ
412
УСТРОЙСТВА, ОДОБРЕННЫЕ RIAA
414
СЕМЬЯ КЕРИ
408
ПУБЛИЧНАЯ БИБЛИОТЕКА
410
КСАН
400
НОКС
402
ИЗГОТОВИТЕЛЬ УСТРОЙСТВА
416
ПК
404
ПОРТАТИВНОЕ УСТРОЙСТВО
406

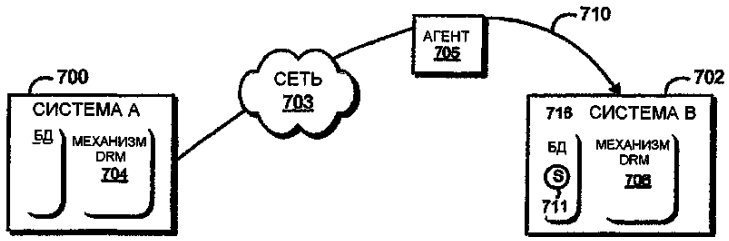
Фиг. 4


НАЧАЛО
ПРИНЯТЬ ЗАПРОС
500
ОЦЕНИТЬ ЛИЦЕНЗИЮ
502
504
НЕТ
АВТОРИЗАЦИЯ?
ДА
508
ОТКЛОНИТЬ ЗАПРОС
506
УДОВЛЕТВОРИТЬ ЗАПРОС
КОНЕЦ

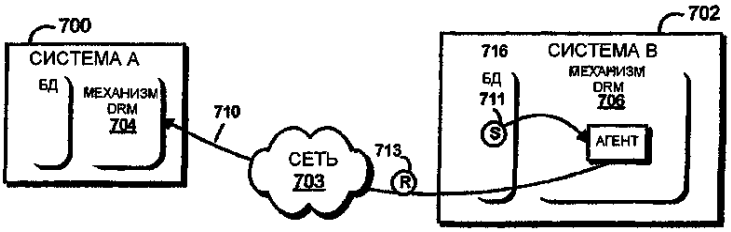
Фиг. 5



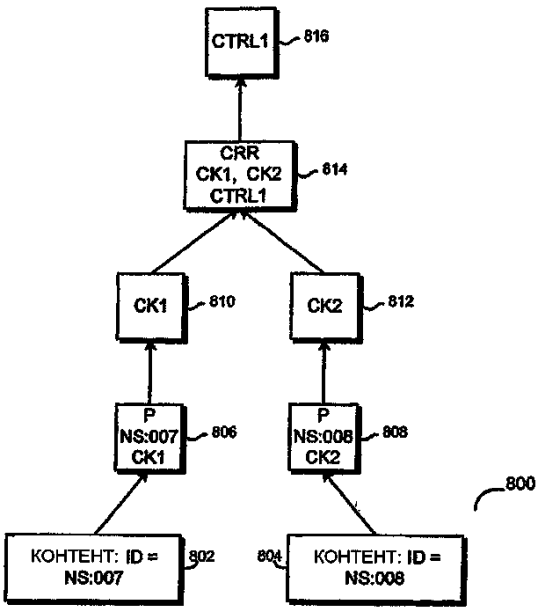
Фиг. 6



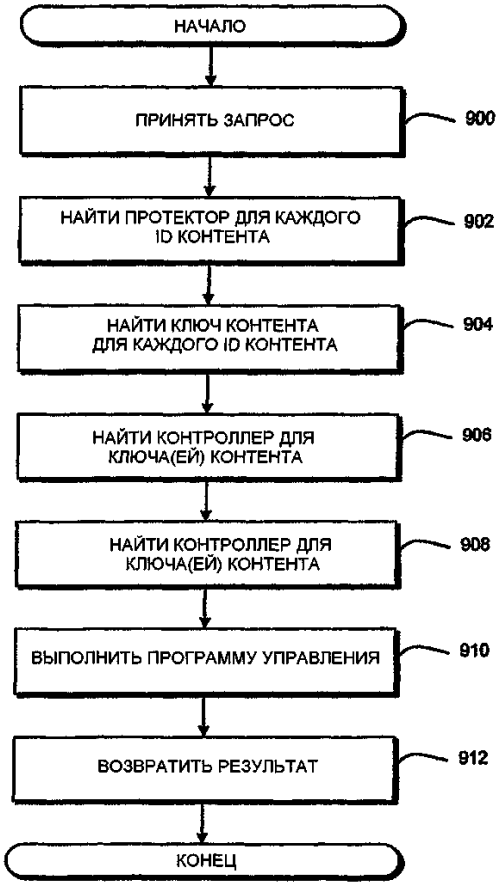
Фиг. 7A



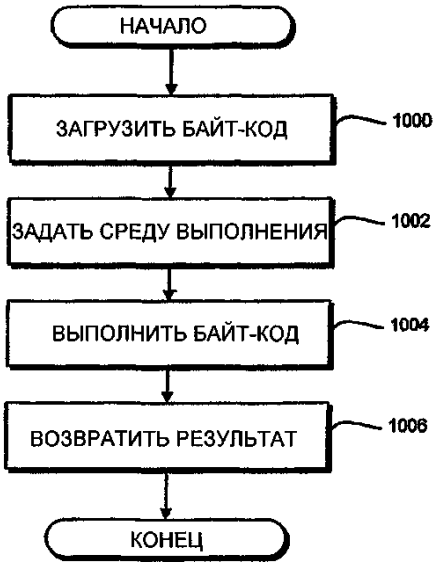
Фиг. 7B



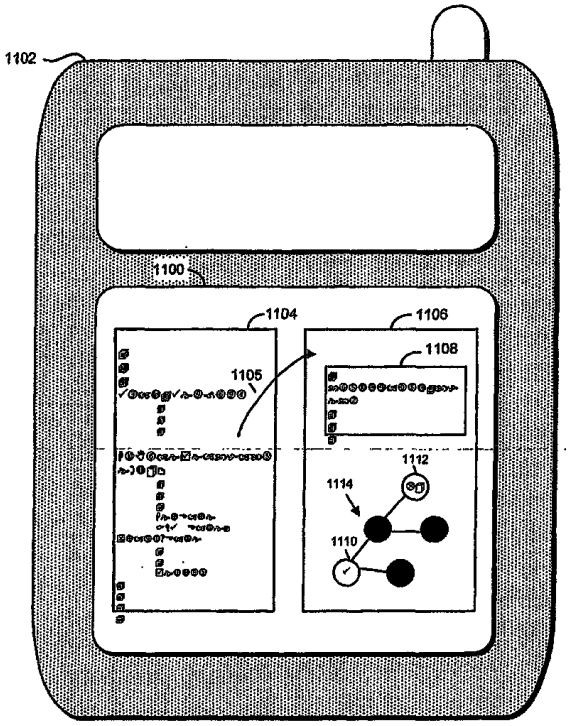
Фиг. 8


НАЧАЛО
ПРИНЯТЬ ЗАПРОС
900
НАЙТИ ПРОТЕКТОР ДЛЯ КАЖДОГО ID КОНТЕНТА
902
НАЙТИ КЛЮЧ КОНТЕНТА ДЛЯ КАЖДОГО ID КОНТЕНТА
904
НАЙТИ КОНТРОЛЛЕР ДЛЯ КЛЮЧА(ЕЙ) КОНТЕНТА
906
НАЙТИ КОНТРОЛЛЕР ДЛЯ КЛЮЧА(ЕЙ) КОНТЕНТА
908
ВЫПОЛНИТЬ ПРОГРАММУ УПРАВЛЕНИЯ
910
ВОЗВРАТИТЬ РЕЗУЛЬТАТ
912
КОНЕЦ

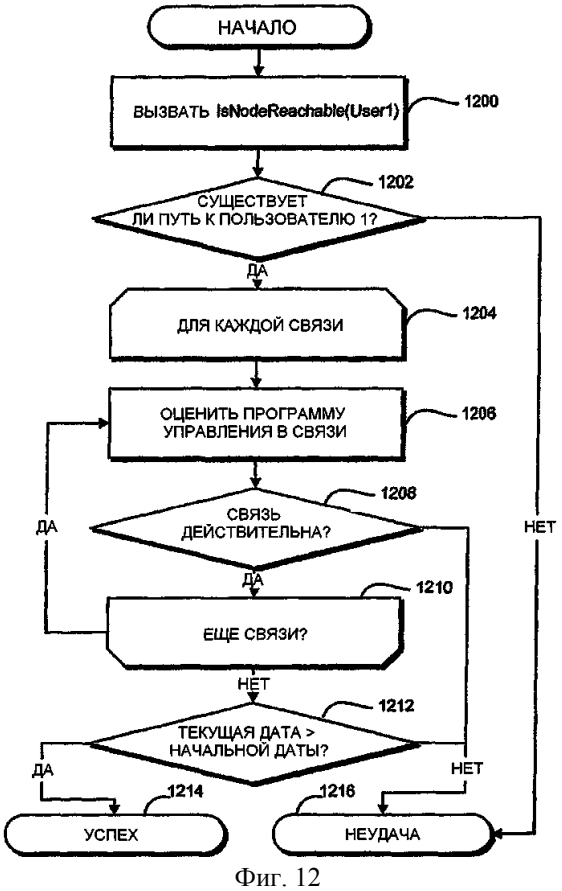
Фиг. 9


НАЧАЛО
ЗАГРУЗИТЬ БАЙТ-КОД
1000
ЗАДАТЬ СРЕДУ ВЫПОЛНЕНИЯ
1002
ВЫПОЛНИТЬ БАЙТ-КОД
1004
ВОЗВРАТИТЬ РЕЗУЛЬТАТ
1006
КОНЕЦ

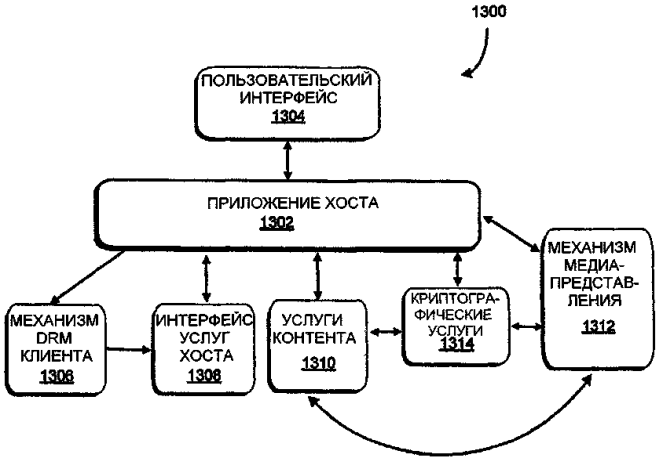
Фиг. 10


1102
1100
1104
1106
1108
1105
1112
1114
1110

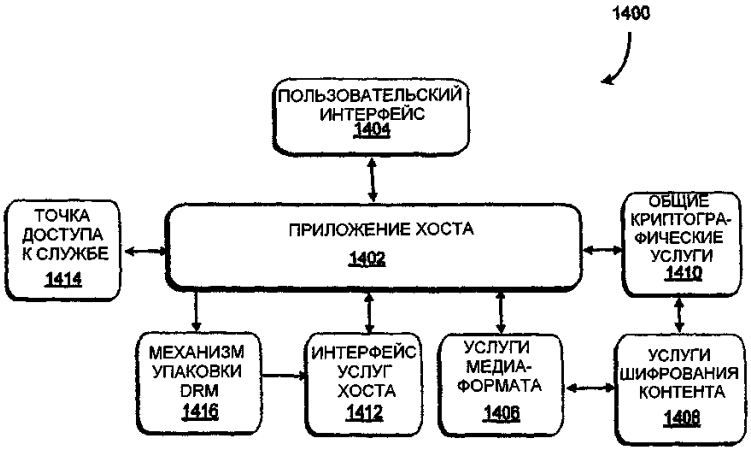
Фиг. 11


НАЧАЛО
ВЫЗВАТЬ IsNodeReachable(User1)
1200
СУЩЕСТВУЕТ ЛИ ПУТЬ К ПОЛЬЗОВАТЕЛЮ 1?
1202
ДА
НЕТ
ДЛЯ КАЖДОЙ СВЯЗИ
1204
ОЦЕНИТЬ ПРОГРАММУ УПРАВЛЕНИЯ В СВЯЗИ
1206
СВЯЗЬ ДЕЙСТВИТЕЛЬНА?
1208
ДА
НЕТ
ЕЩЕ СВЯЗИ?
1210
ДА
НЕТ
ТЕКУЩАЯ ДАТА > НАЧАЛЬНОЙ ДАТЫ?
1212
ДА
НЕТ
УСПЕХ
1214
НЕУДАЧА
1216

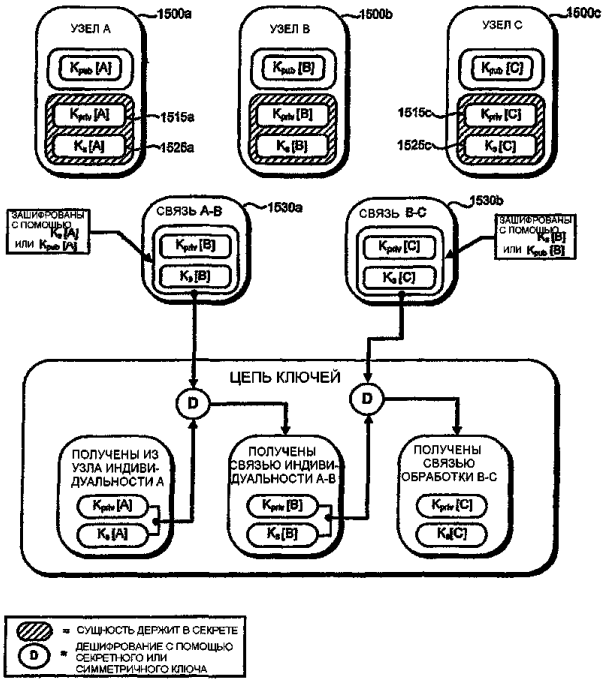
Фиг. 12



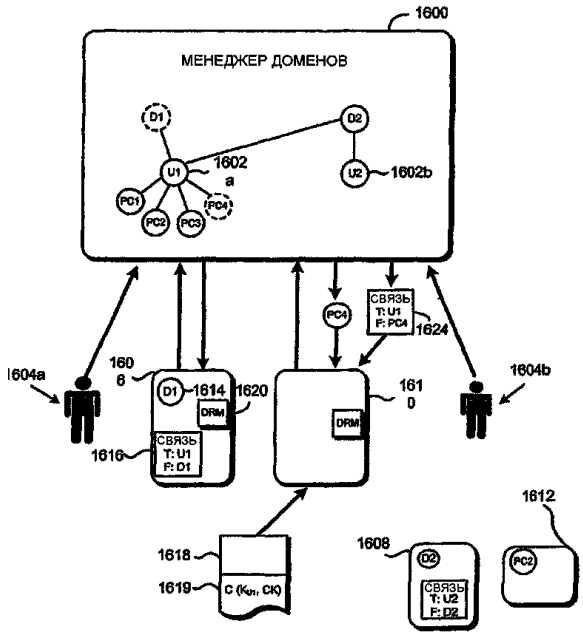
Фиг. 13



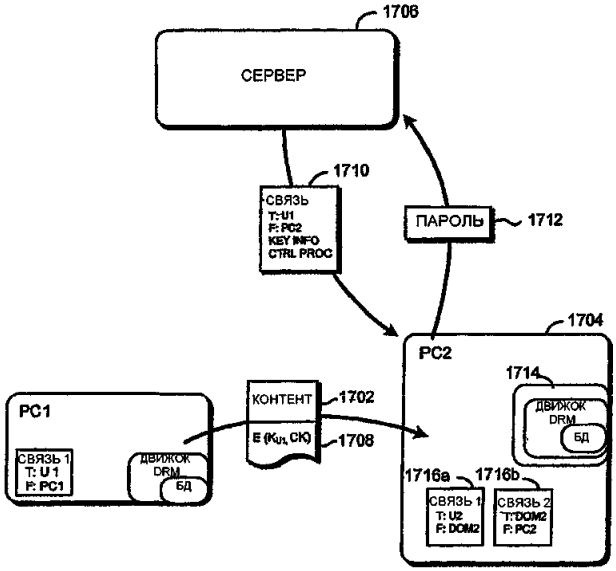
Фиг. 14



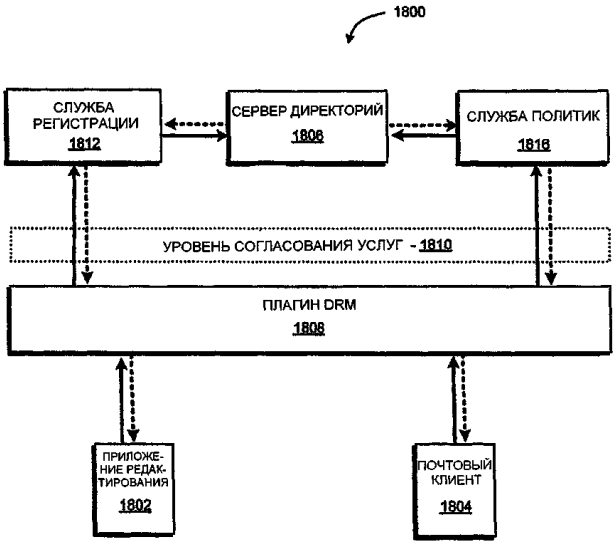
Фиг. 15



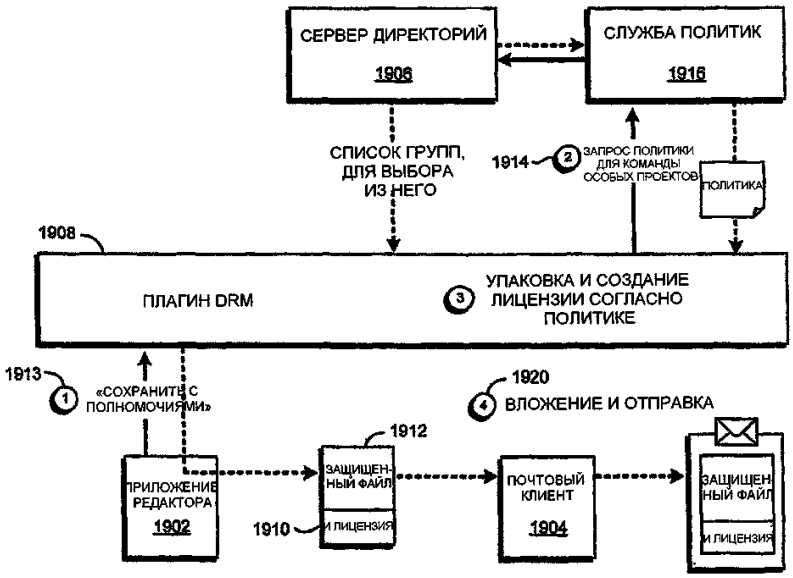
Фиг. 16



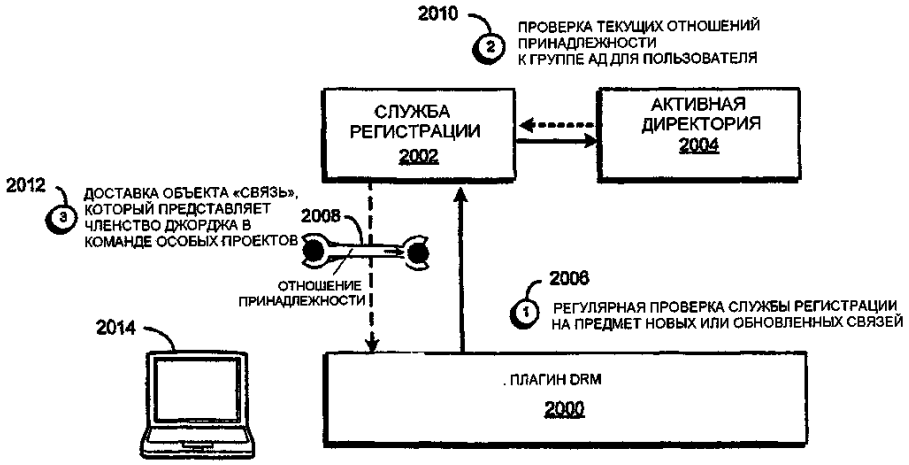
Фиг. 17



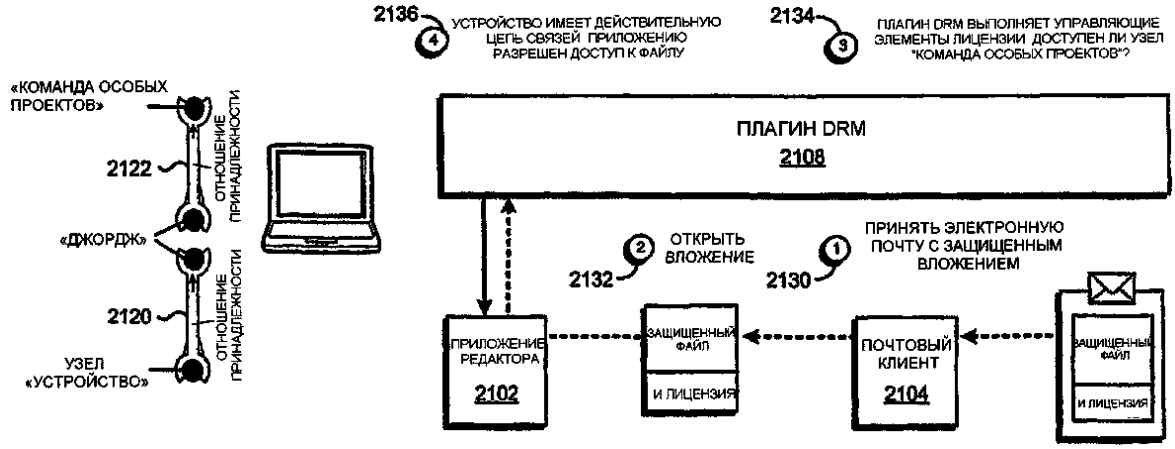
Фиг. 18



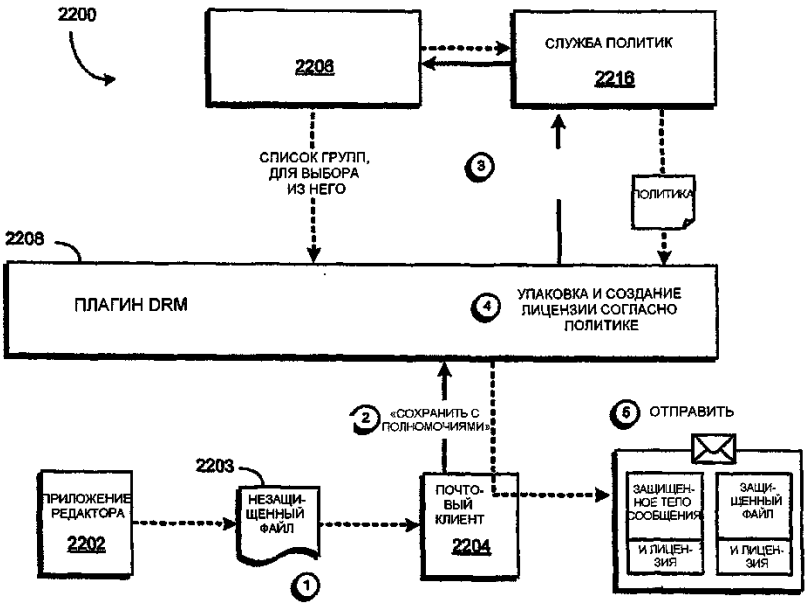
Фиг. 19



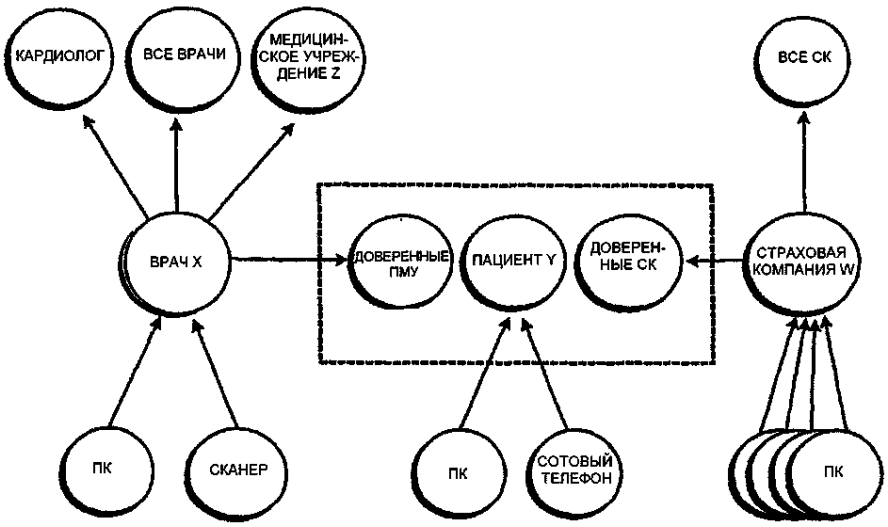
Фиг. 20



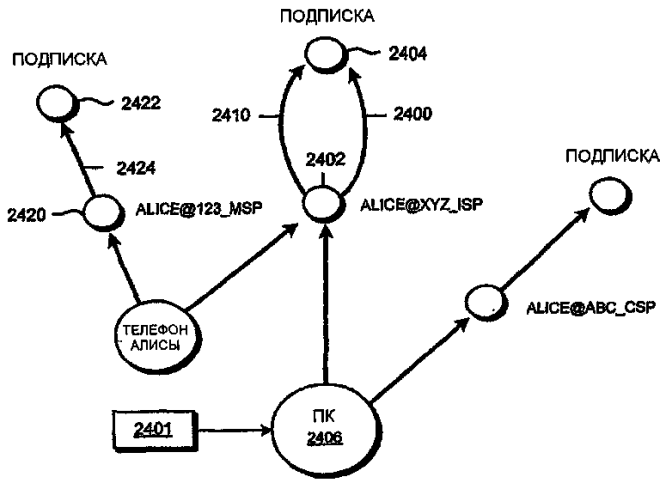
Фиг. 21



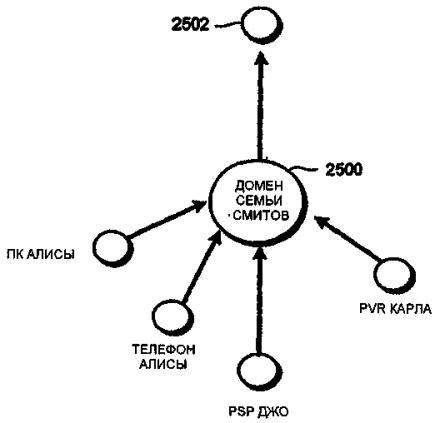
Фиг. 22



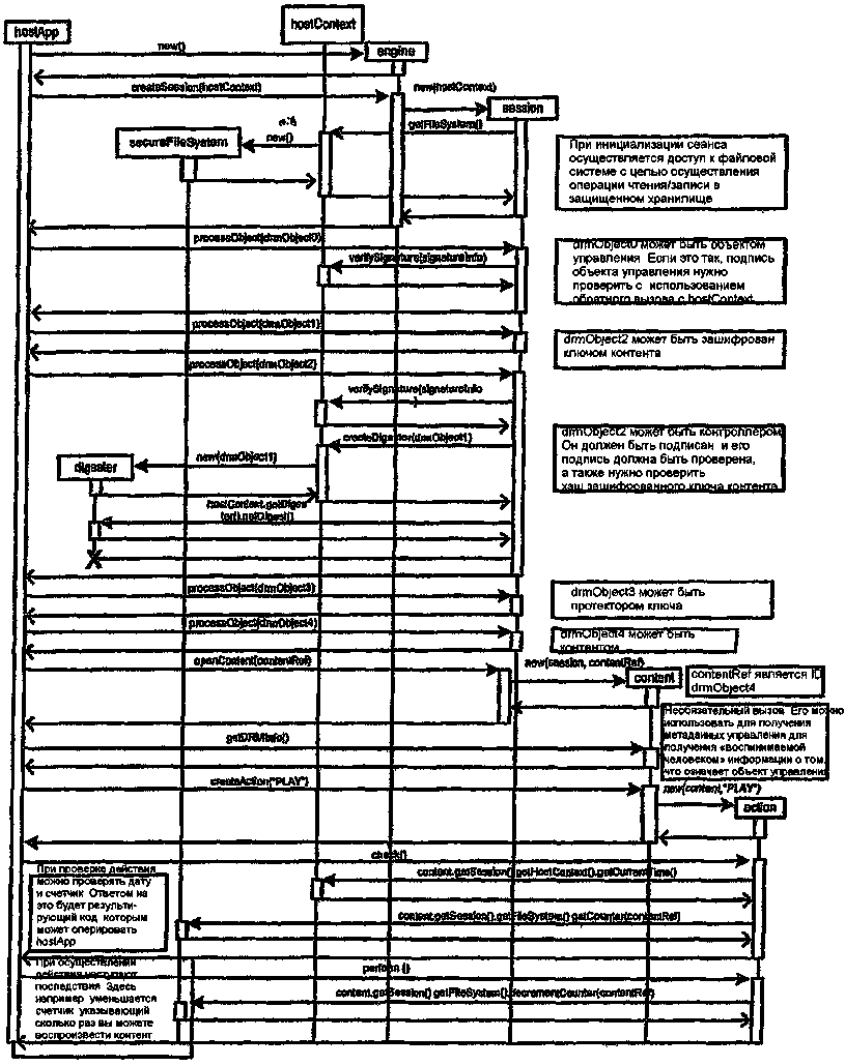
Фиг. 23



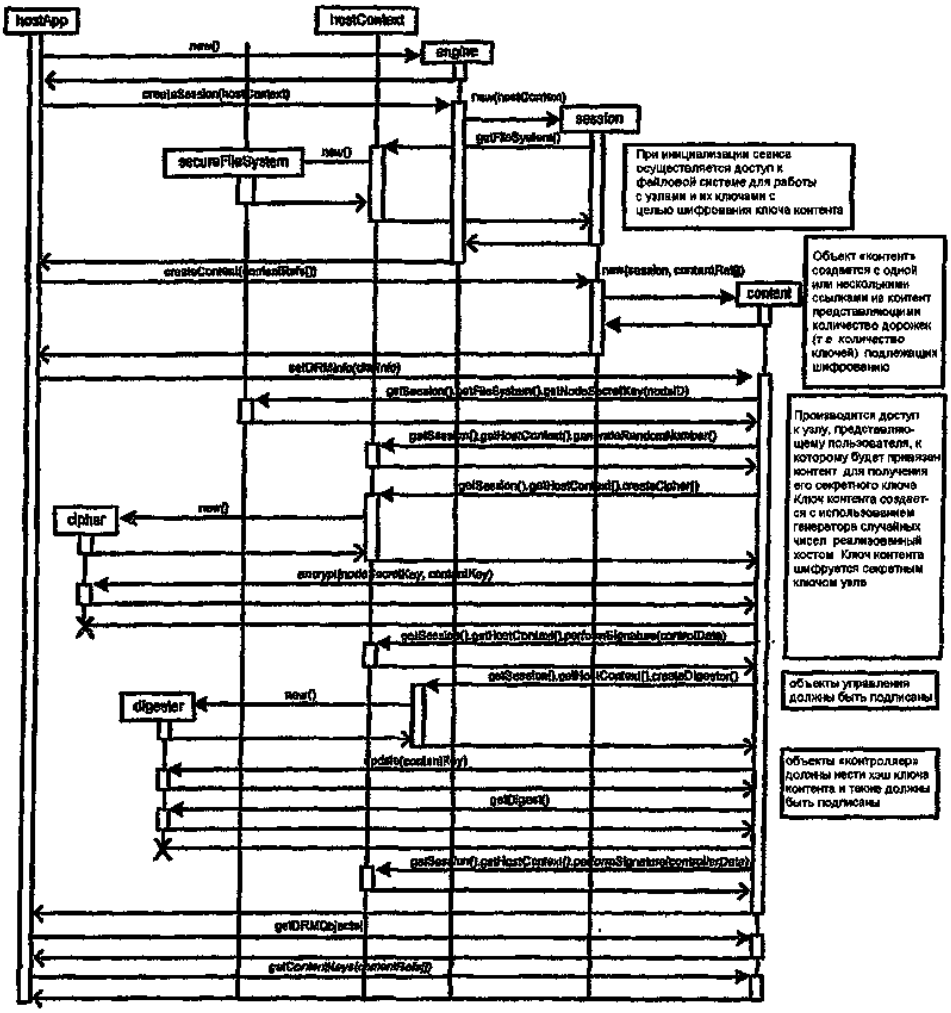
Фиг. 24


2502
2500
ДОМЕН СЕМЬИ СМИТОВ
ПК АЛИСЫ
PVR КАРЛА
ТЕЛЕФОН АЛИСЫ
PSP ДЖО

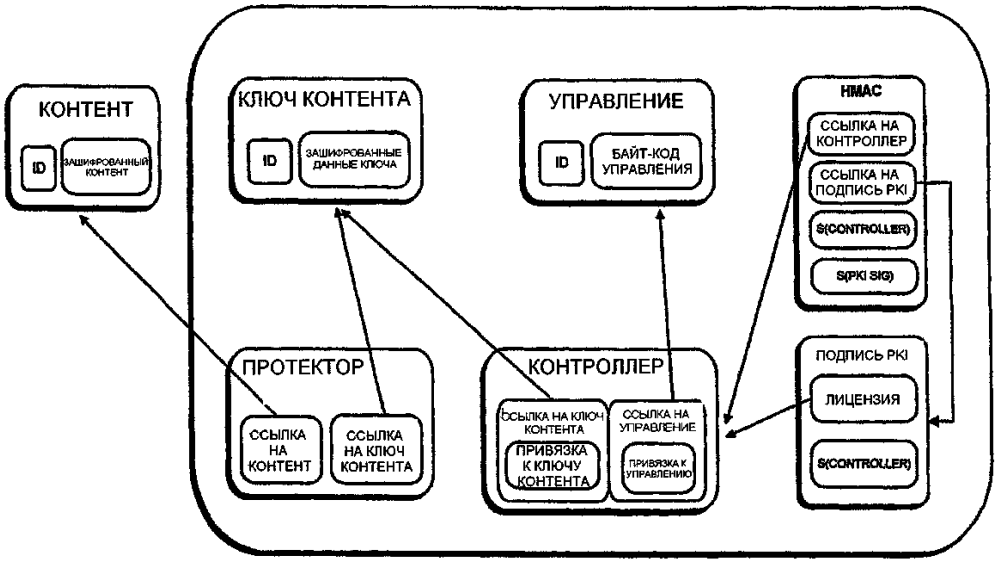
Фиг. 25


hostApp
hostContext
engine
session
securedFileSystem
digester
content
action
При инициализации сеанса осуществляется доступ к файловой системе с целью осуществления операции чтения/записи в защищенном хранилище
drmObject2 может быть зашифрован ключом контента
drmObject3 может быть протектором ключа
drmObject4 может быть контентом
contentRef является ID drmObject4
Необязательный вызов. Его можно использовать для получения метаданных управления для получения «восприимчивой человеком» информации о том, что означает объект управления
При проверке действия можно проверять дату и счетчик. Ответом на это будет результирующий код, которым может оперировать hostApp
При осуществлении действия могут последствия. Здесь например уменьшается счетчик, указывающий сколько раз вы можете воспроизвести контент

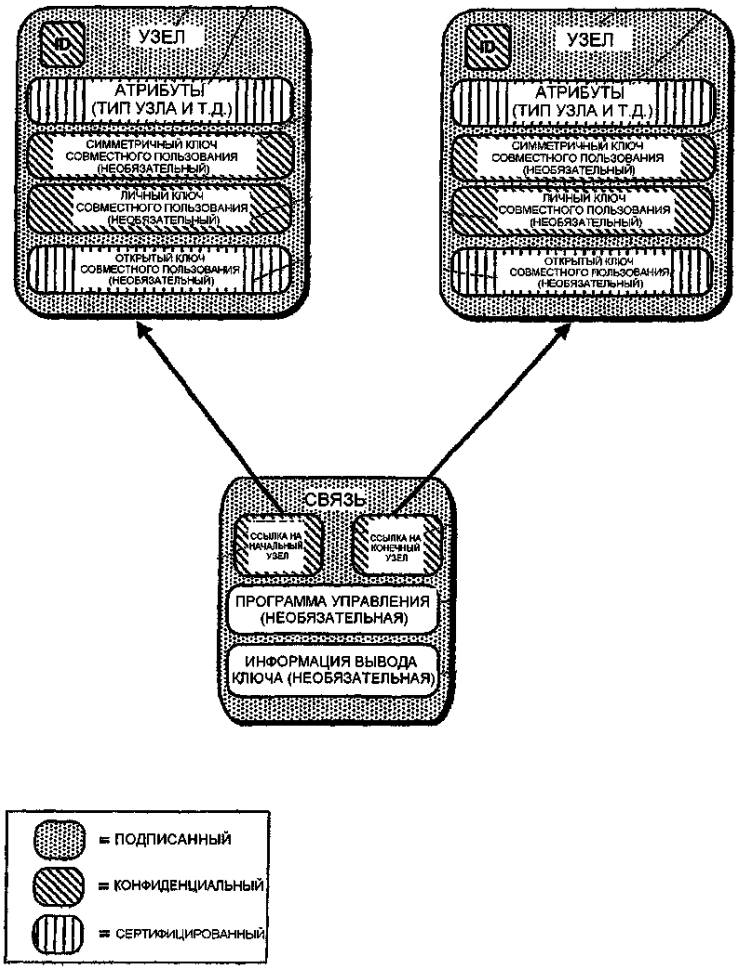
Фиг. 26



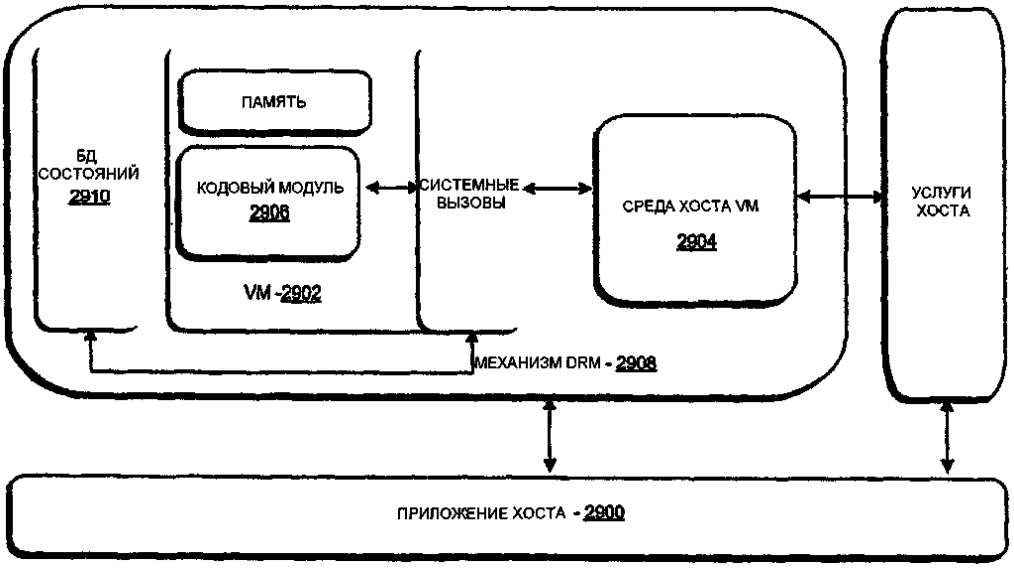
Фиг. 27



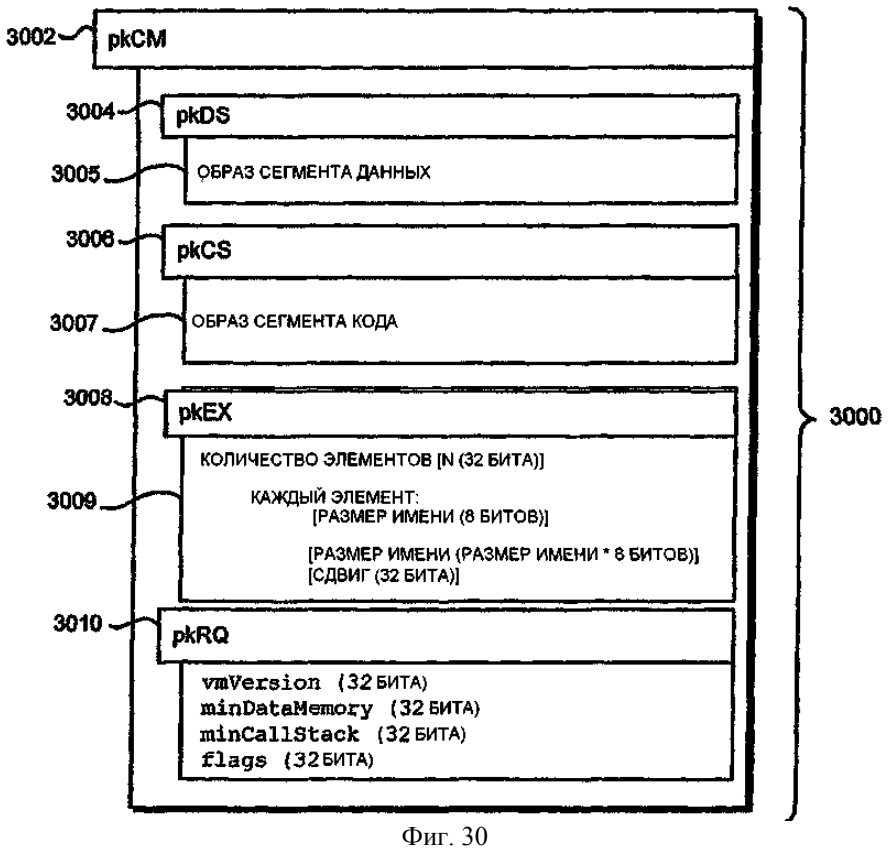
Фиг. 28А



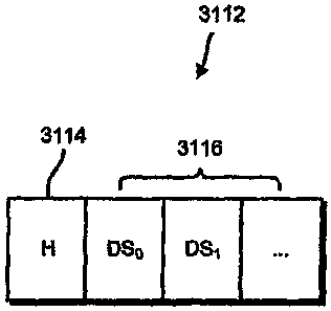
Фиг. 28В



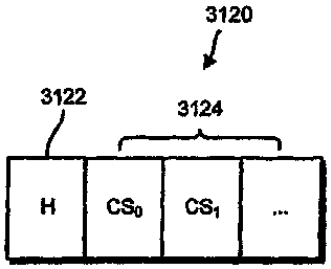
Фиг. 29



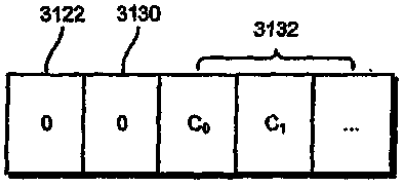
Фиг. 30



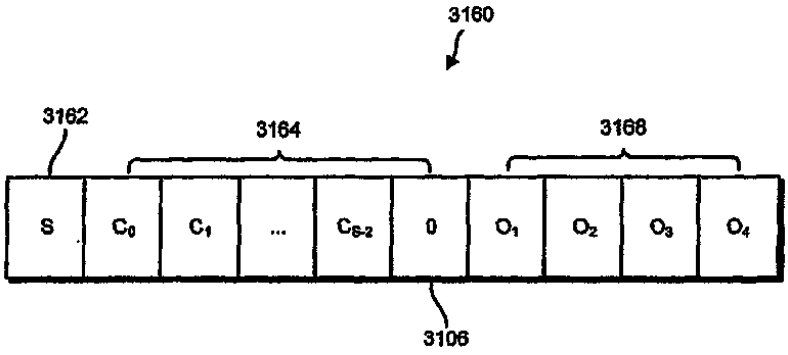
Фиг. 31A



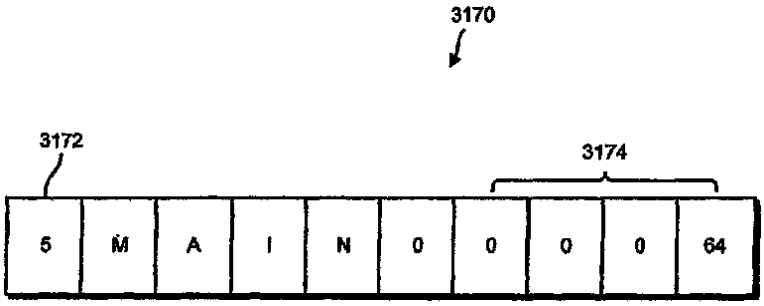
Фиг. 31B



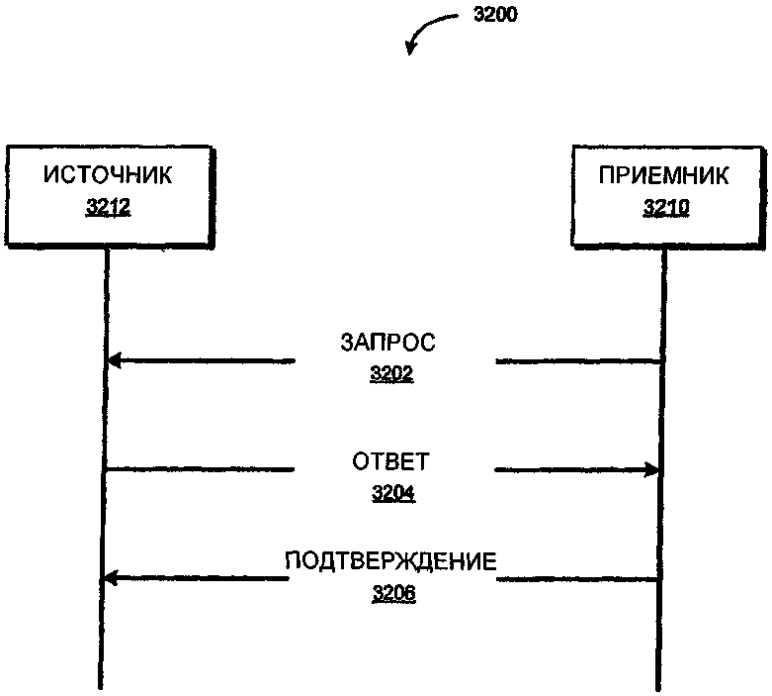
Фиг. 31C



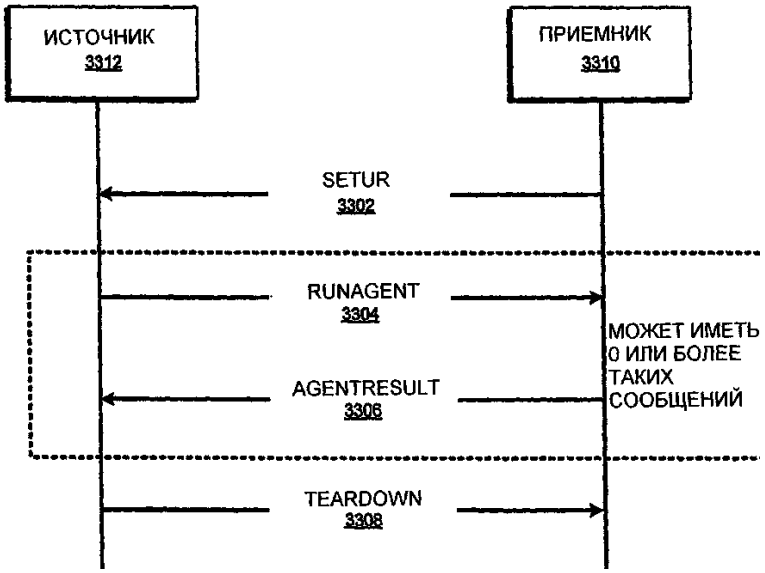
Фиг. 31D



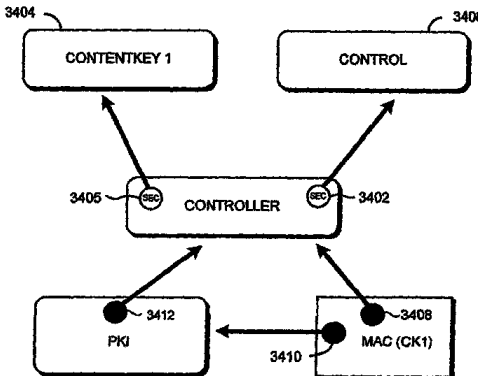
Фиг. 31E



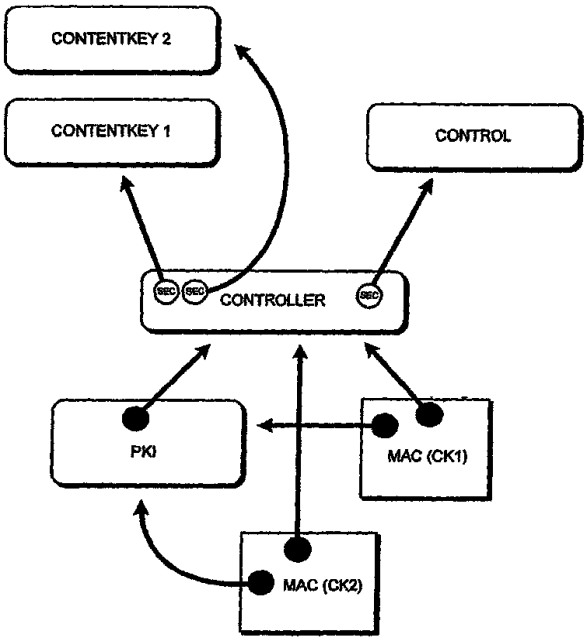
Фиг. 32



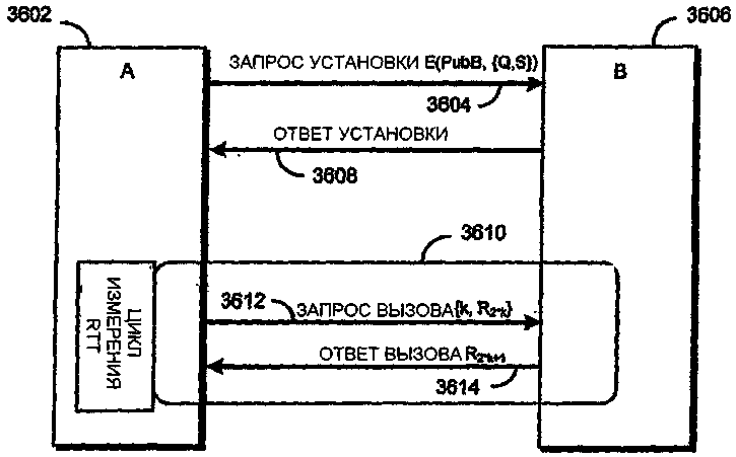
Фиг. 33



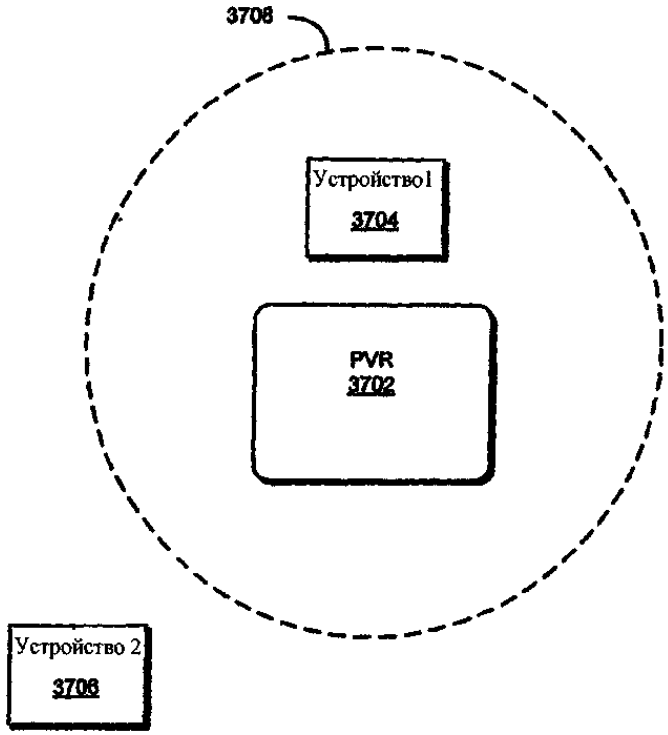
Фиг. 34



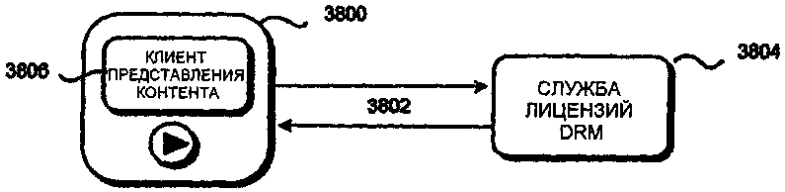
Фиг. 35



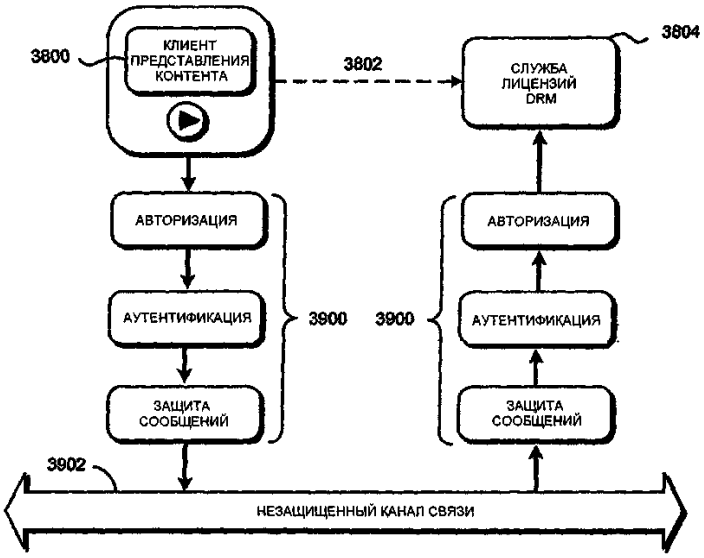
Фиг. 36



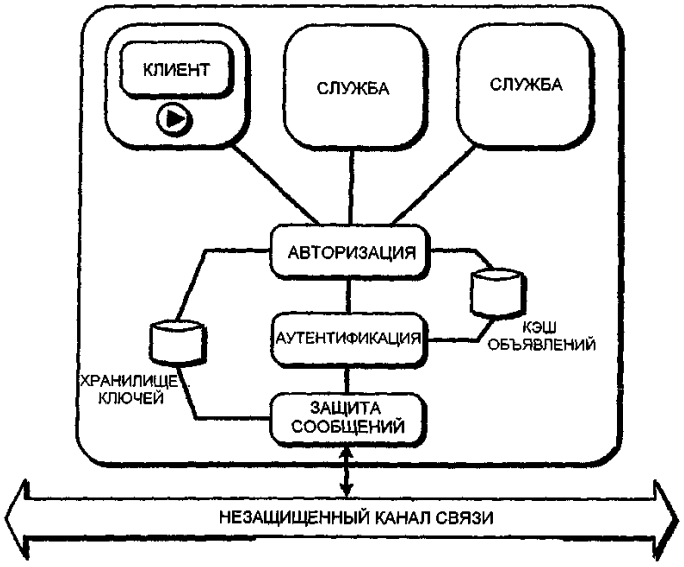
Фиг. 37



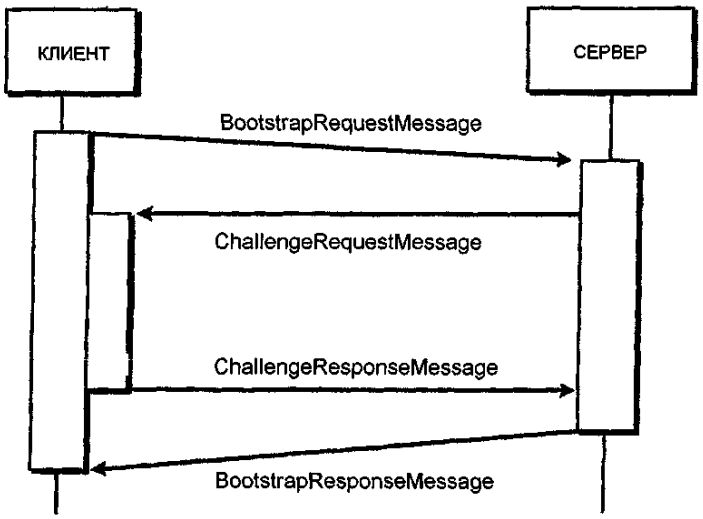
Фиг. 38



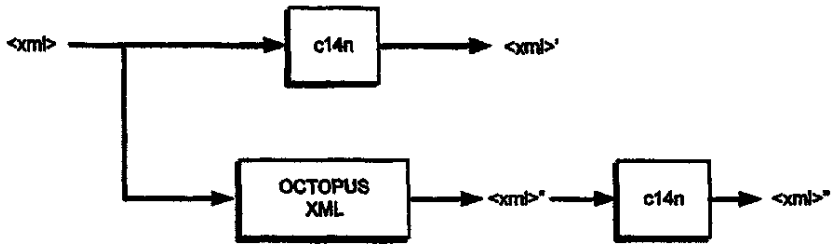
Фиг. 39



Фиг. 40



Фиг. 41



Фиг. 42

Евразийская патентная организация, ЕАПВ
Россия, 109012, Москва, Малый Черкасский пер., 2