(19) Евразийское патентное ведомство

(11) 040905 (13) B1

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ЕВРАЗИЙСКОМУ ПАТЕНТУ

(45) Дата публикации и выдачи патента
2022.08.15

(21) Номер заявки
201990315

(22) Дата подачи заявки
2017.08.31

(51) Int. Cl. ***G06F 12/14*** (2006.01)
G06F 11/30 (2006.01)

(54) ЗАШИФРОВАННЫЙ ТРАНЗИТ И ХРАНЕНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ

(31) **62/395,084**
(32) **2016.09.15**
(33) **US**
(43) **2019.08.30**
(86) **PCT/US2017/049661**
(87) **WO 2018/052726 2018.03.22**

(71)(73) Заявитель и патентовладелец:
НАТС ХОЛДИНГЗ, ЭлЭлСи (US)

(72) Изобретатель:
Аух Юн Хо (US)

(74) Представитель:
Нилова М.И. (RU)

(56) US-A1-20150347480
US-A1-20150378842
US-A1-20160191993
US-A1-20160261413
US-A1-20100174684
US-A1-20140025719

(57) Способ обработки данных включает в себя осуществление доступа, посредством по меньшей мере одного процессора, к единице хранения данных, причем единица хранения данных предоставляет по меньшей мере один входной объект данных и по меньшей мере одну команду превращения, которая должна выполняться по меньшей мере для одного входного объекта данных. По меньшей мере одна команда превращения работает в прямом режиме по меньшей мере для одного входного объекта данных таким образом, чтобы формировать по меньшей мере один выходной объект данных, который должен сохраняться в единице хранения данных.

040905 B1

040905 B1

Перекрестные ссылки на родственные заявки

Заявка на данный патент основана и притязает на приоритет предварительной заявки на патент США № 62/395084, поданной 15 сентября 2016 г., содержимое которой полностью содержится в данном документе по ссылке.

Сущность изобретения

Датацентрическая модель компьютерного проектирования программного обеспечения представляет собой модель, в которой пользовательские данные могут быть приоритезированы выше приложений. Датацентрическое проектирование программного обеспечения может предоставлять возможность защиты данных при хранении. Контейнеризация данных может представлять собой вариант осуществления датацентрического проектирования. Чтобы показывать то, как различные концепты могут реализовываться в рамках этого раскрытия сущности, последовательности чертежей подчеркивают с разных точек зрения конкретные исследуемые концепты, и интегральные чертежи показывают то, как могут взаимодействовать несколько из этих процессов и структур.

Контейнеризация данных может представляться в многоуровневом подходе, и, в предпочтительном случае, каждый уровень может частично или полностью основываться, или работать в сочетании с предыдущими уровнями. Концепты, способы, оборудование, варианты осуществления и/или технические требования, описанные в данном документе для первого уровня, могут совместно называться складыванием структурированных данных с превращениями или SDFT. Концепты, способы, оборудование, варианты осуществления и/или технические требования, описанные в данном документе для второго уровня, который может включать в себя первый уровень, могут совместно называться зашифрованным транзитом и хранением пользовательских данных или NUTS. Любая комбинация каждого уровня может развертываться частично или полностью, чтобы конструировать контейнер для данных, называемый Nut-контейнером, и каждый уровень может развертываться частично или полностью в изоляции. Взаимодействие и/или перемешивание этих двух уровней может быть значительным и/или сложным и может вызывать проблемы для четкого разграничения таких уровней. Следовательно, эти уровни представляются вместе в этом подробном описании. Nut-контейнер может снабжаться различными датацентрическими характеристиками, которые могут обеспечивать логические операции для данных, содержащихся в нем. Для единицы хранения, называемой Nut-контейнер, могут описываться различные варианты осуществления, чтобы показывать то, как определенные общие датаориентированные логические операции могут переопределяться и реструктурироваться, чтобы обеспечивать конфиденциальность, безопасность, удобство и/или возможности пользователей.

Краткое описание чертежей

Различные варианты осуществления могут быть раскрыты в нижеприведенном подробном описании и на прилагаемых чертежах.

Фиг. 1 показывает таблицу символов, используемых для того, чтобы представлять различные типы ключей шифрования.

Фиг. 2 показывает набор упрощенных блок-схем последовательности операций способа, показывающих вводы данных, выводы данных и логические операции, которые типично могут выполняться посредством различных способов шифрования.

Фиг. 3 показывает иллюстрацию общей схемы размещения сети, в которой могут функционировать варианты осуществления этого раскрытия сущности.

Фиг. 4 показывает иллюстрацию вычислительного устройства, в котором могут функционировать варианты осуществления этого раскрытия сущности.

Фиг. 5 показывает иллюстрацию превращения в прямом режиме или в нормальном режиме работы.

Фиг. 6 показывает таблицу общих операций с данными и их классификаций превращений.

Фиг. 7 показывает иллюстрацию превращения в обратном режиме.

Фиг. 8 показывает иллюстрацию необратимого превращения.

Фиг. 9 показывает иллюстрацию условно обратимого превращения.

Фиг. 10 показывает таблицу общих операций с данными и функций, сгруппированных посредством типа превращения.

Фиг. 11 показывает таблицу кодеков, заданных в Python v3.

Фиг. 12 показывает таблицу, перечисляющую дополнительные определения превращения.

Фиг. 13 показывает матрицу обратимости превращений.

Фиг. 14 показывает матрицу модальных действий при превращении.

Фиг. 15 показывает подробный пример превращения с преобразованием в последовательную форму.

Фиг. 16 показывает подробный пример превращения дайджеста.

Фиг. 17 показывает подробный пример превращения дайджеста в обратном режиме, также известного как верификация.

Фиг. 18 показывает иллюстрацию превращения s-шифра.

Фиг. 19 показывает иллюстрацию превращения salsa20 (s-шифра).

Фиг. 20 показывает подробный пример превращения salsa20 (s-шифра).

Фиг. 21 показывает таблицу спецификаций команд для превращений с преобразованием в последовательную форму и сжатием и набора примерных команд превращения, показывающих его использование.

Фиг. 22 показывает таблицу спецификаций команд для превращения с кодированием и набора примерных команд превращения, показывающих его использование.

Фиг. 23 показывает таблицу спецификаций команд для превращения дайджеста и набора примерных команд превращения, показывающих его использование.

Фиг. 24 показывает таблицу спецификаций команд для превращений а-шифра и ц-подписи, и набора примерных команд превращения, показывающих его использование.

Фиг. 25 показывает таблицу спецификаций команд для превращения с извлечением и набора примерных команд превращения, показывающих его использование.

Фиг. 26 показывает таблицу спецификаций команд для превращения 2602 s-шифра и набора 2604 примерных команд превращения, показывающих его использование.

Фиг. 27 показывает формат структуры вывода для выходной строки s-шифра в последовательности из двух этапов, при этом этап 1 иллюстрирует формат ввода, а этап 2 иллюстрирует формат вывода. "Заголовок" представляет собой строку utf8-кодированных параметров ключей/значений переменной длины превращения s-шифра для выходного сообщения.

Фиг. 28 показывает таблицу ключевых слов параметров и технических требований для строки заголовка в формате структуры вывода превращения s-шифра.

Фиг. 29 показывает иллюстрацию итеративных встроенных инкапсуляции сообщений для превращения s-шифра в AEAD-режиме.

Фиг. 30 показывает таблицу спецификаций команд для превращения 3002 замков и набора 3010 примерных команд превращения, показывающих его использование.

Фиг. 31 показывает технические требования различных структур превращений в табличном формате.

Фиг. 32 показывает таблицу спецификаций команд для превращения Мебиуса. Показано его использование, и график показывает структурные изменения, которые он может осуществлять с различными структурами. Матрица показывает допустимые для типа структуры/режима операции, с которыми может работать превращение Мебиуса.

Фиг. 33 показывает таблицу спецификаций команд для превращений 3302, 3304 с прижатием, с очисткой и ключа и набора 3310 примерных команд превращения, показывающих его использование.

Фиг. 34 показывает таблицу для структуры спецификации обмена ключами или KISS.

Фиг. 35 показывает таблицу для KISS-режимов 3502 работы, матрицы 3504, показывающей преобразования типов ключей/формирования полей, и определений 3506 типов ключей.

Фиг. 36 показывает структуру TAR и примеры TAR-определений.

Фиг. 37 показывает блок-схемы, иллюстрирующие то, где сохраняются связанные с превращением атрибуты, и таблицу, перечисляющую тип и местоположения атрибутов.

Фиг. 38 показывает блок-схемы SDFT-операций запутывания и распутывания (или обращения запутывания).

Фиг. 39 показывает блок-схему последовательности операций способа SDFT-операции запутывания.

Фиг. 40 показывает блок-схему последовательности операций способа SDFT-операции распутывания.

Фиг. 41 показывает то, как TAR-обращение выполняется для общей TAR.

Фиг. 42 показывает примеры TAR-обращений.

Фиг. 43 показывает таблицу превращений, преобразованных в шаблон типов ключей, который он может формировать или требовать во время TAR-обработки.

Фиг. 44 показывает TAR-примеры и шаблоны ключей, сформированные из каждого.

Фиг. 45 показывает TAR-примеры и шаблоны ключей, сформированные из каждого, и ожидаемый список KISS-структур, которые должны вводиться (помещаться) или формироваться (формир.). Список KISS также упоминается как стек ключей.

Фиг. 46 показывает три режима работы стека ключей в SDFT TAR-обработке: формирование (формир.), ввод (помещение) и введение (смешанный).

Фиг. 47 показывает иллюстрацию того, как стеки ключей могут формироваться и использоваться в жизненном цикле данных и их TAR.

Фиг. 48 показывает иллюстрацию операций, которые могут возникать в данных, сохраненных в NSstr-структуре.

Фиг. 49 показывает блок-схему последовательности операций способа SDFT-использования, чтобы итеративно складывать данные.

Фиг. 50 показывает блок-схему последовательности операций способа SDFT-использования, чтобы итеративно раскладывать данные.

Фиг. 51 показывает иллюстрацию SDFT-API/библиотеки и различных типов файлов TAR-

определений, к которым она может иметь доступ.

Фиг. 52 показывает примерный Python-сценарий, чтобы выполнять складывание данных вручную.

Фиг. 53 показывает SDFT-пример TAR-определения и его использования в Python-сценарии.

Фиг. 54 показывает блок-схемы динамической TAR-коммутации в одном сеансе связи.

Фиг. 55 показывает блок-схему последовательности операций способа примерного процесса для формирования Nut-идентификатора.

Фиг. 56 показывает блок-схему, показывающую то, где могут использоваться Nut-идентификаторы и идентификаторы замков в Nut-контейнере.

Фиг. 57 показывает примерные взаимосвязи между Nut-идентификаторами, именами путей и рабочими данными.

Фиг. 58 показывает блок-схему варианта осуществления графа Nut-контейнеров или замков, содержащего логические секции: Nut-замок и Nut-части.

Фиг. 59 показывает блок-схему альтернативного варианта осуществления Nut-замка в Nut-контейнере, содержащем три узла замка с замочной скважиной.

Фиг. 60 показывает блок-схему внутренних структур данных узла замка.

Фиг. 61 показывает блок-схему внутренних структур данных секции ввода узла замка, показанного на фиг. 60.

Фиг. 62 показывает диаграмму потоков данных, показывающую взаимосвязь внутренних структур данных замочной скважины под первичный ключ секции ввода, показанной на фиг. 61, когда допустимый первичный ключ может вставляться в замочную скважину.

Фиг. 63 показывает блок-схему последовательности операций способа для процесса вставки ключей для любого узла замка и для любого ключа шифрования.

Фиг. 64 показывает пример, в котором три первичных ключа могут вставляться в замочную скважину под первичный ключ.

Фиг. 65 показывает диаграмму потоков данных операции дешифрования изменяемых замков, продолжающейся из примера, показанного на фиг. 64.

Фиг. 66 показывает диаграмму потоков данных операции шифрования изменяемых замков, продолжающейся из примера, показанного на фиг. 64.

Фиг. 67 показывает таблицу типов изменяемых замков, доступных в любом узле замка, и их характеристик.

Фиг. 68 показывает диаграмму потоков данных операции дешифрования OR-замков.

Фиг. 69 показывает диаграмму потоков данных операции шифрования OR-замков Nut-владельцем.

Фиг. 70 показывает диаграмму потоков данных операции дешифрования MAT-замков.

Фиг. 71 показывает диаграмму потоков данных операции шифрования MAT-замков Nut-владельцем.

Фиг. 72 показывает диаграмму потоков данных операции дешифрования XOR-замков.

Фиг. 73 показывает диаграмму потоков данных операции шифрования XOR-замков Nut-владельцем.

Фиг. 74 показывает диаграмму потоков данных операции дешифрования хеш-замков.

Фиг. 75 показывает диаграмму потоков данных операции шифрования хеш-замков Nut-владельцем.

Фиг. 76 показывает диаграмму потоков данных операции дешифрования SS-замков.

Фиг. 77 показывает диаграмму потоков данных операции шифрования SS-замков Nut-владельцем.

Фиг. 78 показывает блок-схему Nut-контейнера, подчеркивающую ключи слоя.

Фиг. 79 показывает блок-схему последовательности операций способа того, как ключ слоя может вставляться в Nut-контейнере.

Фиг. 80 показывает таблицу разрешений на основе ключей для двух ролей и четырех ролевых игроков.

Фиг. 81 показывает таблицу, перечисляющую множество Nut-частей в примерном Nut-контейнере, в котором каждая часть может представляться посредством узла замка.

Фиг. 82 показывает таблицу, перечисляющую роли доступа на базе разрешений на основе ключей, заданные для типичного Nut-контейнера.

Фиг. 83 показывает блок-схему того, как начальный набор ключей доступа для управления Nut-доступом, называемых ключами отмыкания наборов ключей атрибутов доступа (AAKSUK), может вставляться в замочную скважину под ключ доступа для каждого допустимого первичного ключа.

Фиг. 84 показывает блок-схему распространения NAC-атрибутов доступа из внешних узлов замка во внутренние узлы замка.

Фиг. 85 показывает блок-схему распространения NAC-атрибутов доступа из внешних узлов замка во внутренние узлы замка и вставки выведенного связующего ключа в замочную скважину под первичный ключ связанного узла замка.

Фиг. 86 показывает блок-схему последовательности операций способа для вставки ключей в замочную скважину под ключ доступа.

Фиг. 87 показывает таблицу разрешений на основе ключей для альтернативного варианта осущест-

вления.

Фиг. 88 показывает диаграмму потоков данных для внутренних потоков данных дешифрования узла замка.

Фиг. 89 показывает блок-схему последовательности операций способа для того, чтобы отмыкать Nut-контейнер.

Фиг. 90 показывает блок-схему варианта осуществления системы на основе NUTS и того, как может отмыкаться документ, сохраненный в Nut-контейнере.

Фиг. 91 показывает иллюстрацию широкого использования в NUTS-диалекте, чтобы ссылаться на рабочие данные Nut-контейнера посредством Nut-идентификатора Nut-контейнера, хранящего его. Здесь на ключ шифрования можно ссылаться посредством Nut-идентификатора Nut-контейнера, хранящего его.

Фиг. 92 показывает упрощенный вариант осуществления модели замыкания по списку получателей.

Фиг. 93 показывает упрощенный вариант осуществления модели упорядоченного замыкания.

Фиг. 94 показывает упрощенный вариант осуществления модели упорядоченного замыкания с главным ключом.

Фиг. 95 показывает упрощенный вариант осуществления модели замыкания с главным ключом.

Фиг. 96 показывает упрощенный вариант осуществления модели замыкания с главным ключом.

Фиг. 97 показывает упрощенный вариант осуществления модели замыкания по принципу сейфовой ячейки.

Фиг. 98 показывает упрощенный вариант осуществления модели замыкания по принципу разделения секретов с главным ключом.

Фиг. 99 показывает упрощенный вариант осуществления модели замыкания по принципу PrivaTegrity.

Фиг. 100 показывает упрощенный вариант осуществления конфигурации с несколькими Nut-контейнерами, в которой несколько рабочих данных могут сохраняться в одном Nut-контейнере.

Фиг. 101 показывает упрощенный вариант осуществления конфигурации с несколькими Nut-контейнерами, в которой несколько рабочих данных могут сохраняться в одном Nut-контейнере.

Фиг. 102 показывает упрощенный вариант осуществления модели прямого замыкания с несколькими рабочими данными.

Фиг. 103 показывает упрощенный вариант осуществления упорядоченной пересылки сообщений, демонстрирующей устойчивое к тайным сговорам проектное решение.

Фиг. 104 показывает блок-схему типичных компонентов модульного ввода-вывода.

Фиг. 105 показывает иллюстрацию простых операций чтения и записи с использованием MIOR.

Фиг. 106 показывает трансформации и переносы данных, которые могут быть предусмотрены в типичной операции чтения MIO-файлов.

Фиг. 107 иллюстрирует то, как обратная совместимость форматов файлов может упрощаться с использованием модульного ввода-вывода.

Фиг. 108 иллюстрирует то, как прямая совместимость форматов файлов может упрощаться с использованием модульного ввода-вывода.

Фиг. 109 иллюстрирует то, как модульное отображение может упрощаться с использованием модульного ввода-вывода.

Фиг. 110 иллюстрирует то, как модульное приложение может упрощаться с использованием модульного ввода-вывода.

Фиг. 111 иллюстрирует прогрессивные изменения в Nut-предысторию за два редактирования и в трех точках во времени.

Фиг. 112 иллюстрирует прогрессивные изменения в Nut-журнал регистрации в ходе событий из фиг. 111.

Фиг. 113 показывает то, как ключи на основе взаимосвязей могут представляться в контактных картах Alice и Bob.

Фиг. 114 показывает блок-схему последовательности операций способа того, как спам может обнаруживаться с использованием известных адресов электронной почты и/или RBK.

Фиг. 115 показывает блок-схему последовательности операций способа того, как спам может обнаруживаться с использованием анонимных адресов электронной почты и/или RBK.

Фиг. 116 показывает матрицу состояний на основе детерминированного контекста RBK-канала связи Alice-Bob.

Фиг. 117 показывает матрицу состояний на основе детерминированного контекста RBK-канала связи Alice-производитель.

Фиг. 118 показывает матрицу состояний на основе детерминированного контекста RBK-канала связи производитель-Alice.

Фиг. 119 иллюстрирует изоляцию RBK-взаимосвязей в скомпрометированной системе для Bob.

Фиг. 120 показывает блок-схему предварительно скомплектованных Nut-контейнеров данных.

Фиг. 121 иллюстрирует в виде диаграммы последовательность событий в процессе автоматизиро-

ванной регистрации с использованием RBK.

Фиг. 122 иллюстрирует в виде диаграммы последовательность событий в процессе автоматизированной регистрации с использованием RBK и анонимных адресов электронной почты.

Фиг. 123 показывает таблицу, перечисляющую базовые NUTS-приложения и их описания.

Фиг. 124 показывает блок-схему базовых NUTS-приложений, выполняющихся в компьютерном устройстве.

Фиг. 125 показывает блок-схему NUTserver, выполняющегося в пользовательском устройстве.

Фиг. 126 показывает блок-схему внутренних компонентов, содержащих NUTserver, и их функциональных зависимостей с окружением пользовательского устройства.

Фиг. 127 показывает альтернативный вариант осуществления NUTserver, показанного на фиг. 126, с использованием NoSQL-базы данных в качестве механизма кэширования.

Фиг. 128 показывает блок-схему для схемы размещения серверной MIOR-сети.

Фиг. 129 показывает блок-схему для схемы размещения серверных MIOR-приложений.

Фиг. 130 показывает блок-схему последовательности операций способа для выборки MIO-модулей из MIOR-сервера.

Фиг. 131 показывает блок-схему, иллюстрирующую организацию MIOR-кэша.

Фиг. 132 показывает блок-схему NUTbrowser-приложения в окружении пользовательского устройства.

Фиг. 133 показывает блок-схему NUTbook-приложения в окружении пользовательского устройства.

Фиг. 134 показывает блок-схему инфраструктуры разработки приложений Nut-обработки в окружении пользовательского устройства.

Фиг. 135 показывает блок-схему, иллюстрирующую внутренние компоненты, содержащие NUTbook.

Фиг. 136 показывает блок-схему, иллюстрирующую внутреннюю организацию кэша NUTbook-каталогов из фиг. 135.

Фиг. 137 показывает схему, показывающую организацию иерархических паролей.

Фиг. 138 показывает то, как основной пароль открывает персональный документ согласно иерархическим паролям по фиг. 137.

Фиг. 139 показывает то, как главный пароль открывает персональный документ согласно иерархическим паролям по фиг. 137 и документу на фиг. 138.

Фиг. 140 показывает то, как основные и рабочие пароли открывают рабочий документ согласно иерархическим паролям по фиг. 137.

Фиг. 141 показывает то, как главный пароль открывает рабочий документ согласно иерархическим паролям по фиг. 137 и документу на фиг. 140.

Фиг. 142 показывает блок-схему, иллюстрирующую внутреннюю организацию кэша NUTbook-ключей из фиг. 135.

Фиг. 143 показывает блок-схему последовательности операций способа для того, как NUTbook может просматривать карточный каталог.

Фиг. 144 показывает таблицу услуг на основе NUTS.

Фиг. 145 показывает иллюстрацию схемы размещения сети услуг на основе NUTS.

Фиг. 146 показывает иллюстрацию схемы размещения сети NUTmail-сервера.

Фиг. 147 иллюстрирует в виде диаграммы последовательность событий в процессе автоматизированной регистрации в анонимной почтовой услуге, такой как NUTmail, с использованием RBK.

Фиг. 148 иллюстрирует в виде диаграммы последовательность событий при добавлении канала связи на NUTmail-сервере.

Фиг. 149 иллюстрирует в виде диаграммы последовательность событий, когда Alice и Bob отправляют почтовые сообщения друг другу через NUTmail.

Фиг. 150 показывает иллюстрацию схемы размещения сети NUTchat-сервера.

Фиг. 151 показывает диаграмму потоков данных трех сеансов чата, хостинг которых выполняется посредством NUTchat-сервера.

Фиг. 152 показывает диаграмму потоков данных сохраняемости и репликации предыстории чатов на NUTserver.

Фиг. 153 показывает диаграмму потоков данных для трех отдельных сеансов чата с использованием различных идентификаторов чатов или чат-услуг.

Фиг. 154 показывает диаграмму потоков данных для агностического к тракту диалога, управляемого посредством NUTchat-клиента с использованием трех различных путей чата из фиг. 153.

Фиг. 155 показывает иллюстрацию схемы размещения сети NUTcloud-сервера.

Фиг. 156 показывает иллюстрацию схемы размещения сети NUTnet-сервера.

Фиг. 157 показывает иллюстрацию схемы размещения сети NUThub-сервера для Интернета NUTS (IoN).

Фиг. 158 показывает иллюстрацию прямой сетевой IoN-топологии.

Фиг. 159 показывает иллюстрацию косвенной сетевой IoN-топологии.

Фиг. 160 показывает иллюстрацию NUTserver-концентратора и его соединений с NUThub- и IoN-устройствами из фиг. 159.

Фиг. 161 показывает блок-схему NUThub/IoN-интерфейса в NUTserver-концентраторе из фиг. 160.

Фиг. 162 показывает блок-схему NUThub/NUTserver/IoT-интерфейса в IoN-устройстве из фиг. 160.

Фиг. 163 показывает блок-схему последовательности операций способа для процесса регистрации и конфигурирования для IoN/IoT-устройств.

Фиг. 164 показывает блок-схему последовательности операций способа того, как удаленный управляющий интерфейс может обрабатывать Nut-контейнеры команд из фиг. 161 и 162.

Фиг. 165 показывает иллюстрацию схемы размещения сети NUTS-сервера сертификации.

Фиг. 166 показывает блок-схему, подчеркивающую функциональности NUTS-сервера сертификации из фиг. 165.

Фиг. 167 показывает иллюстрацию схемы размещения сети Wi-Fi/Ethernet-маршрутизатора на основе NUTS.

Фиг. 168 показывает блок-схему последовательности операций способа того, как сообщения могут обрабатываться в Wi-Fi/Ethernet-маршрутизаторе на основе NUTS из фиг. 167.

Фиг. 169 показывает таблицу категорий устройств для Wi-Fi/Ethernet-маршрутизатора на основе NUTS.

Фиг. 170 показывает таблицу атрибутов категории примерного устройства на Wi-Fi/Ethernet-маршрутизаторе на основе NUTS.

Фиг. 171 показывает блок-схему того, как обертывание приложений обеспечивает резервирования и дублирование устройства автоматизированным способом.

Фиг. 172 показывает блок-схему услуги обработки событий (EPS) на двух устройствах.

Фиг. 173 показывает блок-схему типичной настройки сети производителя, которая может использовать отслеживание куки-файлов и предыстории сеансов, сохраненных на серверах обработки больших данных.

Фиг. 174 показывает блок-схему настройки сети производителя, которая может использовать Nut-контейнеры приложений, чтобы записывать копию предыстории сеансов локально, а также сохраненных на серверах обработки больших данных из фиг. 173.

Фиг. 175 показывает блок-схему контекстных вычислений, которые могут осуществляться локально с использованием Nut-контейнера приложения из фиг. 173 и 174.

Фиг. 176 показывает иллюстрацию персональной домашней сетевой топологии, содержащей IoT-устройства и поставщиков услуг.

Фиг. 177 показывает иллюстрацию персональной домашней сети, содержащей два IoN-устройства и их соответствующих поставщиков услуг в косвенной сетевой IoN-топологии, чтобы управлять потоком данных, выходящим к производителям.

Фиг. 178 показывает блок-схему того, как контекстные аналитические приложения могут автоматически фильтровать исходящие IoN-сообщения, чтобы защищать конфиденциальность пользователя в NUTserver из фиг. 177.

Подробное описание изобретения

Оглавление

Символьные обозначения и сокращения.
Шифры и односторонние хеши.
Схема сети.
Схема устройства.
Превращения.
Типы превращений.
Структуры превращений.
Записи с результатами аудита превращений (TAR).
Складывание структурированных данных с превращениями (SDFT).
Nut-идентификатор.
Графы замков и узлы замка.
Замочные скважины.
Изменяемые замки.
Слой.
Управление Nut-доступом (NAC).
Обход узлов замка.
Модульный ввод-вывод.
Чтение и запись.
Обратная совместимость.
Обратная совместимость.
Отображение.
Приложение.

Nut-предыстория.
Nut-журнал регистрации.
Ключи на основе взаимосвязей (RBK).
Анонимные взаимосвязи.
Базовые NUTS-приложения.
NUTserver.
MIOR-сервер.
NUTbrowser/NUTshell.
NUTbook.
Услуги на основе NUTS.
NUTmail.
NUTchat.
NUTcloud.
NUTnet.
NUThub.
NUTS-сервер сертификации.
Wi-Fi/Ethernet-маршрутизатор на основе NUTS.
Обертывание приложений.
Услуга обработки событий.
Контекстные вычисления.
Заключение и философия.

Символьные обозначения и сокращения

Следующие символьные обозначения и сокращения могут использоваться во всех описаниях и на всех чертежах.

Обозначения, помеченные *, могут быть конкретными для NUTS:

AAKS - *набор ключей атрибутов доступа;

AAKSUK - *ключ отмыкания наборов ключей атрибутов доступа;

AAPK - *ключ распространения атрибутов доступа;

а-шифр - асимметричный шифр;

AEAD - аутентифицированное шифрование с ассоциированными данными;

AES - усовершенствованный стандарт шифрования; также Рэндала;

API - интерфейс прикладного программирования;

AKS - *набор ключей доступа;

ARK - *ролевой ключ доступа;

BIOS - базовая система ввода-вывода;

bz2, bzip2 - алгоритм сжатия Барроуза-Уилера;

CA - центр сертификации;

CAC - криптографическое управление доступом;

ChaCha20 - поточный шифр на основе симметричных ключей Бернштейна;

CLI - интерфейс командной строки;

CMAC - код аутентификации сообщений на основе шифра;

Кодек - кодер/декодер; схема кодирования для символьных данных;

COM - объектная модель компонентов;

COR - *класс читателей; или читатель;

CORBA - общая архитектура посредника запросов к объектам;

COW - *класс писателей; или писатель;

CPU - центральный процессор;

CRC - контроль циклическим избыточным кодом;

ц-подпись - *(существительное) цифровая подпись, сформированная с использованием асимметричного закрытого (секретного) ключа;

снабжать ц-подписью - *(глагол), создавать цифровую подпись с использованием асимметричного закрытого ключа;

DK - *извлеченный ключ;

DRM - управление цифровыми правами;

DVD - цифровой видеодиск;

DSA - алгоритм цифровой подписи;

ECC - криптография в эллиптических кривых;

eDK - *зашифрованный извлеченный ключ;

EPS - *услуга обработки событий;

FIPS - федеральные стандарты по обработке информации;

HMAC - хеш-код аутентификации сообщений;

GPS - глобальная система позиционирования;

GPU - графический процессор;
GUI - графический пользовательский интерфейс;
GUID - глобально уникальный идентификатор;
gzip - GNU Zip-сжатие;
HKDF - функция извлечения ключа на основе HMAC;
ikm - материал начального ключа;
IMEI - международный идентификатор мобильного оборудования;
IoN - *Интернет NUTS;
IoT - Интернет вещей;
IPC - межпроцессная связь;
IPv4 - Интернет-протокол версия 4;
IPv6 - Интернет-протокол версия 6;
I/O - Ввод-вывод;
ima - *имя поля KISS, сокращение для "I am a" или "I'm a": определяет KISS-режим;
вект.ин - вектор инициализации: случайное число для криптографического использования;
JSON -система обозначений JavaScript-объектов;
KBP - *разрешения на основе ключей;
Кечака - SHA3-хеш-семейство;
KISS - *структура спецификации обмена ключами;
LAN - локальная вычислительная сеть;
замок - *реализация изменяемых замков в качестве класса превращений;
lzma - цепной алгоритм Лемпела-Зива-Маркова;
MAC - управление доступом к среде (применительно к Ethernet);
MAC - код аутентификации сообщений;
MD5 - дайджест сообщения #5 Ривеста;
MIO - *модульный ввод-вывод;
MIOR - *модульный репозиторий ввода-вывода;
MMS - услуга обмена мультимедийными сообщениями;
NAC - *управление Nut-доступом;
NCS - *NUTS-сервер сертификации;
NFC - связь ближнего радиуса действия;
NIST - Национальный институт стандартов и технологий;
NoSQL - нестандартный язык запросов; также нереляционный стандартный язык запросов;
одноразовый номер - число, используемое только однократно: случайное число для криптографического использования;
NTFS - новая технологическая файловая система (Microsoft);
NUTS - *зашифрованный транзит и хранение пользовательских данных;
OAEP - оптимальное дополнение при асимметричном шифровании Белларе и Рогавея;
ОС - операционная система;
PBKDF2 - функция #2 извлечения ключа на основе пароля посредством RSA (PKCS);
PGP - довольно неплохая конфиденциальность;
PIM - персональный информационный менеджер;
PKCS - криптографические стандарты с общедоступным (открытым) ключом компании RSA Laboratories;
PKCS1_V1.5 - версия 1.5 PKCS #1;
PKI - инфраструктура открытых ключей;
PSS - вероятностная схема подписи;
PUID - практически уникальный идентификатор;
QA - гарантия качества;
QUOPRI - "допущено к печати", или QP-кодирование;
RAM - оперативное запоминающее устройство;
RAT - *уровень корневого доступа, владелец/создатель Nut-контейнера; также RAT-писатель, владелец;
RBAC - управление доступом на основе ролей;
RBCAC - криптографическое управление доступом на основе ролей;
RBK - *ключи на основе взаимосвязей;
ROM - постоянное запоминающее устройство;
RSA - криптосистема с открытым ключом Ривеста-Шамира-Аделмана;
SAC - *управление доступом к слою;
Salsa20 - поточный шифр на основе симметричных ключей Бернштейна;
соль - случайное число для криптографического использования;
s-шифр - симметричный шифр;

SCP - *структурированное криптографическое программирование;
SCRYPT - функция извлечения ключа на основе пароля Персиваля;
SDF - *складывание структурированных данных;
SDFT - *складывание структурированных данных с превращениями;
SHA - защищенный хеш-алгоритм - хеш-разновидность Кечака;
Shake - хеш-разновидность Кечака;
SMS - служба коротких сообщений;
SOAP - простой протокол доступа к объектам;
Спам - незапрошенная массовая почтовая рассылка; также спамовая почтовая рассылка;
SSD - полупроводниковый накопитель;
SSID - идентификатор набора служб;
SSO - единая точка входа;
TAR - архивирование на ленту: Unix-команда, чтобы сохранять данные на ленту или диск;
TAR - *запись с результатами аудита превращений;
TOP - *принцип организации превращений;
зубец - *квота разделения секретов по принципу Шамира, аналогичная зубцам на вилке;
TMX - *превращение;
TOP - *принцип организации превращений;
URL-адрес - универсальный указатель ресурса;
UTF - формат трансформации Unicode;
UTI - универсальный идентификатор типа;
UUID - универсально уникальный идентификатор;
VPN - виртуальная частная сеть;
WAN - глобальная вычислительная сеть;
Wi-Fi - WLAN-протокол;
WLAN - беспроводная LAN;
XML - расширяемый язык разметки;
Zlib - алгоритм Zlib-сжатия.

Фиг. 1 показывает таблицу типов ключей шифрования и их соответствующих символов, которые могут использоваться во всех описаниях и на всех чертежах этого раскрытия сущности. Текстовый пароль или фразовый пароль переменной длины может представляться посредством символа 102. Символ 104 представляет ключ для симметричного шифра, содержащего AES-256 или альтернативный шифр. Символ 106 представляет пару ключей для асимметричного шифра, содержащего RSA-2048 или альтернативный шифр. Открытая часть пары 106 асимметричных ключей может быть проиллюстрирована как символ 108, и закрытая часть может быть показана как символ 110. Специалисты в данной области техники могут легко распознавать, что эти шифры могут представлять собой известные и хорошо протестированные алгоритмы, и что вместо них могут использоваться другие подходящие способы, причем эти способы могут указываться в этом раскрытии сущности, когда может требоваться изменение или альтернатива стандартов.

Шифры и односторонние хеши

Фиг. 2 иллюстрирует базовые операции, которые могут выполняться посредством различных типов шифров. Симметричный шифр 208 в режиме шифрования может подтверждать симметричный ключ 202 и данные 204, чтобы формировать зашифрованные данные 206 или зашифрованный текст. Симметричный шифр 208 в режиме дешифрования может подтверждать идентичный симметричный ключ 202 и зашифрованный текст 206, чтобы формировать исходные данные 204. В реализациях симметричного шифра, способы шифрования и дешифрования могут представлять собой вызовы функций с отдельными именами или могут представлять собой сингулярный вызов с параметром режима в качестве части вводов. Характеристика симметричного шифра может заключаться в том, что процессы шифрования и дешифрования могут использовать идентичный секретный ключ 202.

Асимметричный шифр 214 в режиме шифрования может подтверждать открытую часть пары 210 асимметричных ключей и данные 204, чтобы формировать зашифрованные данные 212 или зашифрованный текст. Асимметричный шифр 214 в режиме дешифрования может подтверждать закрытую часть пары 216 асимметричных ключей и зашифрованный текст 212, чтобы формировать исходные данные 204. В реализациях асимметричного шифра, способы шифрования и дешифрования могут представлять собой вызовы функций с отдельными именами или могут представлять собой сингулярный вызов с параметром режима в качестве части вводов. Характеристика асимметричного шифра может заключаться в том, что процессы шифрования и дешифрования могут использовать различные части пары ключей. В такой реализации, как RSA-2048, открытый ключ может извлекаться из закрытого ключа с использованием математической взаимосвязи, в силу чего закрытый RSA-2048-ключ может быть синонимичен с парой ключей, и открытый ключ может извлекаться из нее.

Способ 222 на основе цифровой подписи в режиме подписания может подтверждать закрытую часть пары 216 асимметричных ключей и зашифрованный текст 218, чтобы формировать цифровую под-

пись 220. Способ 222 на основе цифровой подписи в режиме аутентификации может подтверждать открытую часть пары 210 асимметричных ключей, цифровую подпись 220 и зашифрованный текст 218, чтобы аутентифицировать 224 то, создана или нет цифровая подпись с использованием упомянутого зашифрованного текста 218 и закрытой части пары 216 асимметричных ключей. В реализациях способа на основе цифровой подписи, способы подписания и аутентификации могут представлять собой вызовы функций с отдельными именами или могут представлять собой сингулярный вызов с параметром режима в качестве части вводов. Характеристика способа на основе цифровой подписи может заключаться в том, что процессы подписания и аутентификации могут использовать различные части пары ключей. В такой реализации, как способ на основе цифровой подписи на основе пар RSA-2048-ключей, открытый ключ может извлекаться из закрытого ключа с использованием математической взаимосвязи, в силу чего закрытый RSA-2048-ключ может быть синонимичен с парой ключей, и открытый ключ может извлекаться из нее. Для краткости и краткости, этот документ может взаимозаменяемо упоминать цифровую подпись как "ц-подпись"; этап снабжения цифровой подписью фрагмента данных может взаимозаменяемо упоминаться как "снабжение ц-подписью"; снабжение цифровой подписью фрагмента данных может взаимозаменяемо упоминаться как "снабженный ц-подписью".

Способ на основе цифровой подписи может представлять собой тип кода аутентификации сообщений или MAC. MAC могут создаваться с односторонними хеш-алгоритмами для данных. Хеш-способ, такой как SHA512, может подтверждать контент данных таким образом, чтобы формировать его дайджест сообщения, который может иметь длину вплоть до 512 битов. Аутентификация MAC с использованием таких способов, как SHA512, влечет за собой повторное вычисление MAC для упомянутого фрагмента данных и сравнение предоставленного MAC и вычисленного MAC на предмет равенства. Технология, известная как код аутентификации сообщений хеширования по ключу или HMAC, может принимать дополнительный ввод криптографического ключа наряду с контентом данных таким образом, чтобы формировать HMAC-значение.

Способы на основе цифровой подписи и/или способы хеширования могут использоваться в различных частях этого раскрытия сущности, чтобы формировать дайджесты сообщений, которые могут представлять соответствующие данные.

Схема сети

Фиг. 3 представляет упрощенную схему сети, в которой различные варианты осуществления этого раскрытия сущности могут применяться частично или полностью. Глобальная вычислительная сеть WAN 302 может представляться как сетевое облако, которое может содержать множество серверов, маршрутизаторов и коммутационных систем в различных центрах связи, работающих совместно с возможностью предоставлять упрощенный вид сетевого облака пользователю или компании. Облачные услуги 304 также могут быть графически упрощены в качестве облака, которое представляет различные коммерческие системы, которые могут предоставлять сетевые услуги, такие как облачное хранилище данных и облачная обработка; эти услуги могут реализовываться как собственные технические требования, но могут содержать множество экземпляров ферм серверов, массивов хранения данных и маршрутизаторов. Персональный маршрутизатор 306 может соединяться с WAN 302, который может предоставлять отдельным пользователям доступ в Интернет между множеством соединимых сетевых устройств 308-318, которые может иметь пользователь. Пользователь может не быть ограничен устройствами, проиллюстрированными на схеме, но может использовать любое устройство, которое может использовать маршрутизатор для того, чтобы осуществлять доступ к другим устройствам в идентичной сети или в Интернете. Маршрутизатор 306 может представлять собой выделенное устройство маршрутизации для поставщика Интернет-услуг, либо он может представлять собой комбинированное устройство, предоставляющее маршрутизацию и/или поддержку LAN и/или WLAN, и может упоминаться как шлюз. Корпоративный маршрутизатор 320 может соединяться с WAN 302, которая может предоставлять институциональным пользователям доступ в Интернет между множеством соединимых сетевых устройств 302-330, которые может иметь компания. Компания может не быть ограничена устройствами, проиллюстрированными на схеме, но может использовать любое устройство, которое может использовать маршрутизатор для того, чтобы осуществлять доступ к другим устройствам в идентичной сети или в Интернете. Маршрутизатор 320 может представлять собой выделенное устройство маршрутизации для поставщика Интернет-услуг, либо он может представлять собой набор взаимно соединенных и управляемых маршрутизаторов, предоставляющих маршрутизацию и/или поддержку LAN и/или WLAN, и может упоминаться как шлюз и/или сеть intranet. Система и способ, описанные в данном документе в различных вариантах осуществления, могут использоваться и применяться к некоторым или всем частям этой схемы сети.

Схема устройства

Общее вычислительное устройство 400 проиллюстрировано на фиг. 4. Блок 404 обработки может соединяться с системной шиной 402, которая может упрощать часть или всю внутреннюю связь и переносы данных в устройстве. Может быть предусмотрено несколько различных типов доступных системных шин, но для простоты они могут совместно называться системной шиной 402. Блок обработки может представлять одно- или многоядерный процессор, а также матрицы процессоров, аналогичные матрицам процессоров, содержащимся в различных специализированных платах обработки, таких как GPU-

платы и компактные тонкие серверы. Другие компоненты, обслуживаемые посредством системной шины, могут представлять собой сетевые адаптеры 412; интерфейсы 410 ввода-вывода; интерфейсы 414 отображения; постоянное запоминающее устройство ROM 406, которое может сохранять BIOS-программу 408; энергозависимое запоминающее устройство RAM 416, которое может эфемерно сохранять рабочую операционную систему 418, работающие приложения 420 и/или данные 422 приложений; и энергонезависимое запоминающее устройство 424, такое как накопители на жестких дисках, SSD и флэш-накопители 426, которые совместно могут постоянно сохранять установленную операционную систему 428, приложения 430 и/или файлы 432 данных.

Не все компоненты проиллюстрированного вычислительного устройства могут требоваться для обеспечения применимости или функциональности некоторых или всех вариантов осуществления этого раскрытия сущности. Например, устройства могут не иметь ни физических дисплеев, ни интерфейсов ввода-вывода, аналогичных дисплеям и интерфейсам, содержащимся в некоторых IoT-устройствах; маршрутизаторы и шлюзы могут иметь очень слабые ресурсы в отношении физических жестких дисков. Обязательное требование для поддержки NUTS и совместимости может заключаться в способности выполнять NUTS-совместимое программное обеспечение, которое может содержать блок обработки, некоторую форму хранилища и системную шину.

Превращения

Превращения могут представлять собой предпочтительный способ организации множества известных операций манипулирования данными, содержащихся в компьютерном программировании. NUTS может обозначать означенное в качестве принципа организации превращений или TOP. Кроме того, любая систематическая операция манипулирования данных может анализироваться с использованием TOP и может классифицироваться в качестве типа превращения. После этого, превращение может категоризироваться, нормализовываться, структурироваться, интегрироваться и/или адаптироваться с возможностью работать совместно в рамках инфраструктуры TOP, что может называться складыванием структурированных данных с превращениями или SDFT. Информативные перспективы TOP и/или работы с данными с помощью SDFT могут предоставлять возможность реализации лучших и/или сложных проектных решений данных концептуально более простым и/или программно-эффективным способом. TOP и SDFT могут представлять собой предпочтительные низкоуровневые механизмы реализации для NUTS-компонентов.

Анализ, способы и/или структуры на основе превращения данных могут показывать то, как разделение таких концептов на уровни и проектирование их ассоциированных способов позволяет задавать реализуемый набор интегрированных структур данных и алгоритмических способов, которые могут предоставлять возможность легко достижимых и систематических превращений данных модульным, портативным, допускающим хранение и/или самоописываемым способом. Вследствие многоуровневого и переплетающегося характера такого анализа, описания превращений могут иметь прямые и обратные ссылки и могут требовать от читателя обращаться к различным разделам для того, чтобы получать лучшую оценку определенных характеристик. Складывание структурированных данных с превращениями (SDFT) основывается на превращениях с использованием структур и технологий передачи данных и может помогать обеспечивать возможность хранения, возможность передачи, модульность, портативность, инкапсулируемость и/или временную совместимость превращенных данных.

В проектном NUTS-решении SDFT представляет собой набор низкоуровневых операций и может считаться фундаментальным компоновочным блоком, чтобы более легко конструировать Nut-контейнер. Тем не менее, SDFT может использоваться независимо, частично или полностью для того, чтобы упрощать определенные трудоемкие и/или повторяющиеся превращения данных в приложении. SDFT может обеспечивать возможность протоколам компьютерной связи динамически коммутировать последовательности превращений и/или параметрические разбросы превращений в идентичном сеансе между двумя различными приложениями. В настоящее время, такая односеансная динамическая коммутация может представлять собой нетривиальное программируемое упражнение. Использование SDFT для того, чтобы компоновать Nut-контейнер, может не представлять собой обязательное требование, но его признаки могут помогать компоновать Nut-контейнер более целесообразно, понятно и гибко. SDFT дополнительно может описываться в качестве технологии на основе перехода состояния данных, которая обеспечивает бесконечные варьирования событий перехода с четко определенными поведениями относительно обратимости последовательностей перехода состояния, и может предоставлять итеративную технологию инкапсуляции, чтобы сохранять необходимые атрибуты и данные простым контекстно-зависимым способом. SDFT подтверждает и охватывает множество каждодневных сложностей при программировании и может представлять прагматичный набор принципов организации, в которых теоретические подтверждения могут подчиняться эмпирическим подтверждениям.

Фиг. 5 показывает то, как принцип организации превращений может рассматривать любую операцию с данными в качестве превращения 510 данных, которое может требовать источника 502 входных данных и атрибутов 504 и которое может выводить превращенные данные 512 и ассоциированные атрибуты 514. Любое четко определенное манипулирование, операция, преобразование и/или трансформация данных могут классифицироваться в качестве типа превращения 510. TOP может обеспечивать возмож-

ность систематически конструировать согласованный набор способов превращения данных модульным, портативным и/или самоописываемым способом.

Таблица на фиг. 6 показывает примерный набор общих операций с данными и то, как они могут классифицироваться с использованием TOP. Превращения могут охватывать класс фундаментальных операций с данными, которые традиционно могут сегрегироваться в восприятии и на практике. Это может иметь место, когда программисты обсуждают криптографию и сжатия данных, эти два класса операций с данными типично могут рассматриваться как две сильно отделенные и отличающиеся операции для данных. Помимо алгоритмических различий каждой операции, с точки зрения TOP, эти операции могут рассматриваться в качестве типа превращения с шифрованием и превращения со сжатием. В таблице, "JSON-преобразование в последовательную форму" может классифицироваться в качестве превращения "с преобразованием в последовательную форму" с операцией "json", в силу чего выполняемая команда превращения может объявляться как "serialize json". Вызов AES-шифрования с использованием симметричного шифра для фрагмента данных может классифицироваться в качестве превращения "s-шифра" с операцией "aes", в силу чего выполняемая команда превращения может объявляться как "sci-pher aes". Специалисты в данной области техники могут легко распознавать все остальные типы операций с данными, перечисленных в таблице, и придерживаться шаблона организации классификации превращений и категоризации операций.

Фиг. 7 показывает схему превращения в обратном режиме или операции обращения. Этот чертеж является идентичным фиг. 5, за исключением стрелок потоков данных, которые протекают в противоположном направлении. Превращение 510 может иметь четко определенный обратимый алгоритм, как проиллюстрировано посредством блока 710. Обратимое превращение 710 может требовать в качестве ввода источник 712 превращенных данных и атрибуты 714 и может выводить исходные данные 702 и ассоциированные атрибуты 704. Может существовать область техники вычислений, называемая обратимыми вычислениями, которая может демонстрировать концепты, аналогичные концептам обратимого превращения. Могут возникать некоторые различия в целях каждого принципа организации. Обратимые вычисления могут теоретизировать существование обобщенного языка обратимых вычислений, операции которого могут реализовываться вниз вплоть до кремниевых уровней для возможной эффективности использования энергии общих вычислений. Обратимые превращения могут быть нацелены на конкретную реализацию TOP для таких преимуществ, как, но не только, минимизация написанного кода, минимизация программируемых ошибок, удобное управление ключами, упрощение формирования ключей, структурирование самоописываемых данных, нормализация концептов манипулирования данными, введение независимых от языка программирования способов выполнения превращений и/или упрощение компоновки сложных структур криптографических данных.

Фиг. 8 показывает графическое представление необратимого превращения. Превращение 810 в прямом режиме может выполнять превращение для данных 802 и атрибутов 804, которые могут формировать превращенные данные 812 и атрибуты 814, но эти выводы наряду с типом манипулирований, которые превращение может выполнять для вводов, могут иметь необратимый характер. Такие необратимые превращения могут примерно иллюстрироваться посредством хешей, MAC, сжатий данных с потерями и других односторонних функций или манипулирований данными. TOP может вводить технологии анализа, которые могут периферийно дополнять характеристики таких необратимых превращений, и может формировать операции, которые могут задавать их характеристики обратного превращения.

Фиг. 9 показывает блок-схему условно обратимого превращения. Такой превращение 910 может иметь четко определенный обратимый алгоритм, но может требовать дополнительных вводов и/или атрибутов 914 для успешного выполнения операции обращения. Условно обратимое превращение 910 может требовать в качестве ввода источник 912 превращенных данных и/или атрибуты 914 и может выводить исходные данные 902 и/или ассоциированные атрибуты 904, если и когда требуемые условия 916 удовлетворяются, в противном случае, оно может завершаться сбоем с состоянием 920 ошибки. Шифры, которые могут требовать ключей, могут классифицироваться в качестве условно обратимых превращений, поскольку отсутствие корректного ключа (атрибута) может затруднять дешифрование зашифрованного текста.

Фиг. 10 является таблицей, перечисляющей общие операции с данными и функции и их соответствующие классификациям превращений. Специалисты в данной области техники могут распознавать некоторые или все операции с данными и/или функции, перечисленные в таблице. В примерных целях, материал, представленный в этом документе, может ссылаться на язык программирования, называемый Python версия 3.6, и его синтаксис, концепты, функцию и/или способы. Множество криптографических функций, упоминаемых в этом раскрытии сущности, могут содержаться в Python Standard-, pyCrypto-dome-, secretsharing- и/или pyCrypto-библиотеках. Специалисты в данной области техники могут находить эквивалентный синтаксис, концепты, функцию и/или способы в большинстве современных языков программирования и в их соответствующих библиотеках. Следует отметить, что "ц-подпись" представляет собой превращение для цифровых подписей; другая мнемоника может быть перечислена в разделе "Символьные обозначения и сокращения" этого документа. Дополнительные подробные описания каждого превращения могут содержаться в таблице на фиг. 12.

Фиг. 11 является таблицей кодеков, заданных в Python v3.6. Этот список может быть неполным вследствие множества существующих кодеков, пролиферации новых кодеков и/или ограничений, заданных в Python v3.6. "Кодек" является сокращением для "кодирование/декодирование" и представляет собой преобразование для набора символов в вычислении. Символу назначается произвольное, но уникальное двоичное значение в рамках одного кодека; в силу этого полный набор символов может задавать один кодек. Не все символы в данном кодеке могут быть человекочитаемыми или печатаемыми. Широкое использование кодеков служит для надлежащего представления различных наборов символов различных языков. Превращение "с кодированием" может позволять выполнять любую из этих операций кодирования кодека для означенной строки данных. Кодеки с названиями, начинающимися с "utf ", могут указывать кодеки, соответствующие формату трансформации Unicode (UTF), который может представлять собой принцип организации международных наборов символов для множества стандартов на основе Интернета.

Фиг. 12 является таблицей, перечисляющей превращения, поясненные выше, с их подробными описаниями. Дополнительные превращения могут задаваться в рамках инфраструктуры с использованием TOP, как показано посредством последних шести превращений в таблице: ключей, с очисткой, TAR-групп, с прижатием, замков и Мебиуса. Некоторые из этих дополнительных превращений могут быть конкретными для библиотеки складывания структурированных данных с превращениями (SDFT), некоторые могут быть конкретными для языка, и/или некоторые могут представлять собой операции, связанные с NUTS. Это может иллюстрировать гибкий характер TOP посредством предоставления возможности задания и классификации новых типов превращений, чтобы расширять их репертуар. Этот гибкий признак расширения обусловлен проектным решением таким образом, что SDFT-библиотека может приспосабливать новые операции превращения в будущем. Признак расширения также может предоставлять возможность добавления "задним числом" устаревших версий операций превращения для обратной совместимости. Преимущество такой гибкости может представлять собой способность SDFT-обработанных данных получать характеристику лучшей временной совместимости. Временная совместимость данных может задаваться как те характеристики, которые могут обеспечивать возможность простой обработки сохраненных данных посредством идентичного или другого приложения в некоторый будущий момент времени. Временные несовместимости могут представлять собой результат, но не только, различий версий форматов файлов приложений, различных кодирований символов, устаревших форматов файлов приложений, различий способов работы с данными, различий в упорядочении операций с данными и/или конкретных для операций с данными параметрических разбросов.

Фиг. 13 показывает матрицу обратимости превращений. Каждое превращение может обозначаться как обратимое, необратимое и/или условно обратимое. Критерии для задания такого обозначения могут быть основаны на идеальных намерениях превращения, а не на его реализованных и/или теоретических недостатках на практике. В других случаях, обозначение может быть произвольным. Это может быть проиллюстрировано посредством операции дайджеста, называемой MD4, которая может формировать хеши с длиной в 128 битов исходных данных. MD4-хеш может считаться очень слабым алгоритмом хеширования по сравнению с такой операцией хеширования, как 512-битовый SHA2, вследствие ее чувствительности к коллизиям, что могут быть нежелательной чертой в алгоритмах хеширования. Перспектива TOP может заключаться в том, чтобы распознавать одно из исходных намерений MD4 в качестве необратимого уникального хеша и категоризировать его этим способом. Такая категоризация может не исключать этот тип превращения из получения четко определенной, инжиниринговой характеристики обратимости через дополнительный TOP-анализ, как показано в последующем разделе. Превращение со сжатием может попадать в рамки как обратимых, так и необратимых обозначений на основе конкретной выполняемой операции сжатия. Множество технологий сжатия изображений и/или аудио могут демонстрировать необратимость вследствие их характера дискретизации; 12-мегабайтное цифровое изображение может сжиматься до 360 Кбайт для эффективной передачи через приложение чата, но вследствие характера человеческого визуального восприятия, общее впечатление от изображения может надлежащим образом передаваться, несмотря на безвозвратные потери данных. Такой сжатое изображение может необратимо модифицироваться вследствие огромного объема исходных данных, которые могут отбрасываться во время трансформации.

Обратимое превращение со сжатием может примерно иллюстрироваться посредством gzip-сжатия; оно может работать по принципу идентификации и уменьшения повторяющихся битовых комбинаций в двоичных данных, но оно может сохранять достаточно информации для того, чтобы обращать процесс и воспроизводить исходные данные полностью. Условно обратимое превращение может примерно иллюстрироваться посредством симметричного AES-шифра; оно может работать по принципу вовлечения открытого текста и симметричного ключа и формирования зашифрованного текста. Процесс дешифрования может принимать ключ и зашифрованный текст, чтобы формировать исходный открытый текст. Таким образом, представление корректного симметричного ключа для зашифрованного текста может представлять собой обязательное условие, которое должно удовлетворяться, чтобы дешифровать зашифрованный текст или обращать процесс шифрования.

TOP может задавать режим превращений, который может указывать направление данной операции

превращения как прямое или обратное. Прямой режим превращения может выполнять свой нормальный процесс и/или свой инжиниринговый прямой процесс. Обратный режим превращения может выполнять свой внутренне присущий обратный процесс и/или свой инжиниринговый обратный процесс. Таблица на фиг. 14 показывает матрицу, указывающую тип операции, которую превращение может выполнять внутренне на основе своего режима превращений. В качестве ссылки, таблица перечисляет общеизвестные названия операций, такие как "преобразование в последовательную форму" и "отмена преобразования в последовательную форму" либо "шифрование" и "дешифрование". Следует отметить инжиниринговые обратные процессы дайджеста и ц-подписи: "дайджест" и "верификация", "подписывание" и "аутентификация". Для превращения с "очисткой", при котором он может удалять различные внутренние данные, ассоциированные с его структурой данных превращений, может быть невозможным восстанавливать такие удаленные данные без надлежащих дополнительных данных и/или повторного выполнения процесса прямого превращения для исходных данных, чтобы воспроизводить удаленные переходные данные. Превращение "ключей" может выполнять операции формирования и/или управления ключами, связанные с выполнением превращений. В связи с этим, вследствие внутренне присущего случайного характера формирования ключей, может быть невозможным теоретически и/или алгоритмически обращать такой процесс детерминированным способом за конечный промежуток времени. В последующем разделе подробно поясняется аспект управления ключами для превращения "ключей", когда рассматривается то, как превращения могут работать в контексте складывания структурированных данных (SDF); в аспекте управления ключами для превращения ключей может быть затруднительным проектировать обратимый дубликат вследствие его характеристики установления надлежащих структур ключа для успешной обработки в SDF-контексте.

Фиг. 15 показывает три последовательные схемы, каждая из которых дополнительно детализирует пример превращения 1504 с преобразованием в последовательную форму, который может быть обозначен как обратимое превращение. В компьютерном программировании, технология преобразования в последовательную форму и/или маршалинга может принимать сложную конкретную для языка внутреннюю структуру данных и может систематически деконструировать и/или организовывать ее контент линейно, чтобы формировать эквивалентную строку данных или набор строк данных (далее называется строкой данных). Форма строки данных может быть более подходящей для долговременного хранения, передачи данных и/или дополнительных превращений. Преобразование в последовательную форму по определению может требовать его полной обратимости логическим способом, с тем чтобы реконструировать исходный контент на инициирующем языке или его эквиваленте. Python-структура 1512 может превращаться с использованием JSON-операции 1514, чтобы формировать эквивалентную JSON-строку 1516, и обратный процесс может быть возможным, как показано посредством двунаправленных стрелок последовательности операций обработки. Простая древовидная структура данных показана на 1522, которая может примерно иллюстрировать сложную Python-структуру данных. Превращение 1524 с преобразованием в последовательную форму может формировать эквивалентную строку 1526 из 1522. Эта выходная строка 1526 теперь может сохраняться, передаваться и/или превращаться по мере выполнения программы.

Фиг. 16 показывает три последовательные схемы, каждая из которых дополнительно детализирует пример превращения 1606 дайджеста, который может быть обозначен необратимое превращение. Этот пример показывает операцию SHA2-хеширования в качестве превращения 1616 дайджеста, которое может требовать в качестве вводов данные 1612 и длину 1614 дайджеста в качестве атрибута 1604. Превращение 1616 дайджеста с SHA2-хешированием может формировать хеш указанной длины 1618. Python-строка 1622 данных и требуемая длина 1624 дайджеста 256 битов могут представлять собой вводы в превращение 1626 с SHA2-хешированием, чтобы формировать хеш-строку 1628 с длиной в 256 битов.

Фиг. 17 показывает подробный пример превращения дайджеста в обратном режиме, также известного как верификация. В TOP, оно может упоминаться как инжиниринговое обращение превращения. Превращение 1710 дайджеста может подтверждать в качестве вводов данные D1 1702 и атрибуты A1 1704, чтобы выполнять превращение 1710 дайджеста в прямом режиме, которое может формировать в качестве вывода 1708 строку DG1 1712 дайджеста. Обратный режим этого превращения 1720 может подтверждать в качестве вводов данные D1 1722 1736 года, атрибуты A1 1724 и строку 1728 дайджеста DG1, чтобы выполнять превращение 1720 дайджеста в обратном режиме, которое может формировать в качестве вывода 1738 флаг и/или значение, указывающее то, строка 1728 дайджеста DG1 верифицирована 1732, или верификация завершена сбоем 1734. Процесс 1740 верификации может формировать строку 1722 дайджеста DG2 посредством выполнения превращения 1720 дайджеста в прямом режиме для вводов D1 1722 и A1 1724. Выходная строка 1722 дайджеста DG2 затем может сравниваться на предмет равенства 1730 относительно входной строки 1728 дайджеста DG1. Результат сравнения 1738 может представляться в некоторой форме, чтобы показывать то, завершено или нет успешно обратное превращение дайджеста. Таким образом, разработка этого обращения дайджеста может требовать повторной обработки прямого режима превращения и сравнения выводов вместо базирования на обходных путях касательно нахождения логической обратимости таких операций, которые могут быть затруднительными, длительными и/или недостижимыми.

Фиг. 18, 19 и 20 показывают подробный пример превращения s-шифра в прямом и обратном режиме, также известного как симметричный шифр. Превращение 1806 s-шифра в прямом режиме может подтверждать в качестве ввода открытый текст 1802 и атрибуты 1804, чтобы формировать в качестве вывода зашифрованный текст 1810 и/или атрибуты 1812. Превращение 1826 s-шифра в обратном режиме может подтверждать в качестве ввода зашифрованный текст 1830 и атрибуты 1832, чтобы формировать в качестве вывода открытый текст 1822 и/или атрибуты 1824. Фиг. 19 иллюстрирует то, как симметричный salsa20-шифр может работать в качестве превращения s-шифра. Превращение 1906 s-шифра salsa20 в прямом режиме может подтверждать в качестве ввода открытый текст 1902 и атрибуты 1904, чтобы формировать в качестве вывода зашифрованный текст 1910. Атрибуты для этого конкретного шифра в прямом режиме могут содержать двоичный симметричный ключ, длину ключа и/или значение соли. Эта прямая salsa20-реализация (шифрование) может представлять собой потоковый шифр, и дополнительные выходные атрибуты не могут формироваться во время процесса шифрования, за исключением любых скрытых атрибутов, которые функция может встраивать в собственной выходной строке. Превращение 1926 s-шифра salsa20 в обратном режиме может подтверждать в качестве ввода зашифрованный текст 1930 и атрибуты 1932, чтобы формировать в качестве вывода открытый текст 1922. Атрибуты для этого конкретного шифра в обратном режиме могут содержать двоичный симметричный ключ, длину ключа и/или значение соли. Эта обратная salsa20-реализация (дешифрование) может представлять собой потоковый шифр, и дополнительные выходные атрибуты не могут формироваться во время процесса дешифрования. Фиг. 20 иллюстрирует то, как симметричный salsa20-шифр может работать в качестве превращения для дискретизированных данных, что может представляться в рамках библиотечной Python v3.6- и PyCryptodome-функции. Превращение 2006 s-шифра salsa20 в прямом режиме может подтверждать в качестве ввода строку данных 2002 и атрибуты 2004, содержащие 256-битовый симметричный секретный ключ и одноразовый номер (в качестве соли), чтобы формировать в качестве вывода зашифрованный текст 2010. В Python, симметричный ключ может представляться как "байтовая" строка типа данных, и в силу этого атрибут длины ключа может легко извлекаться посредством функции len() для байтовой строки ключа. Превращение 2026 s-шифра salsa20 в обратном режиме может подтверждать в качестве ввода зашифрованный текст 2030 и атрибуты 2032, содержащие 256-битовый симметричный секретный ключ и одноразовый номер, чтобы формировать в качестве вывода открытый текст 2022. В Python, симметричный ключ может представляться как "байтовая" строка типа данных, и в силу этого атрибут длины ключа может легко извлекаться посредством функции len() для байтовой строки ключа. Атрибуты 2032, возможно, должны быть эквивалентными атрибутам 2004 для этого условно обратимого превращения, чтобы надлежащим образом обрабатывать зашифрованный текст 2030 в обратном режиме (дешифровать), чтобы восстанавливать исходный открытый текст 2002.

Типы превращений

В следующих таблицах и примерах, представленных на фиг. 21-35, каждое превращение может не быть ограничено операциями, указываемыми в этой таблице; любая подходящая операция может анализироваться через TOP и затем может интегрироваться в инфраструктуру, чтобы расширять функциональные характеристики конкретного превращения. Синтаксис и конструкты Python v3.6 могут использоваться для того, чтобы подробнее иллюстрировать примеры. Эквивалентные типы данных, структуры, синтаксис и/или способы могут содержаться и подставляться на различных языках программирования специалистами в данной области техники. В некоторых случаях, опция ключ/значение может не быть релевантной для конкретного языка или библиотеки, и она может игнорироваться или модифицироваться по мере необходимости при условии, что обработка может приводить к эквивалентным результатам.

Превращение с преобразованием в последовательную форму/сжатием

Фиг. 21 показывает таблицу спецификаций команд для превращений 2102 с преобразованием в последовательную форму и сжатием и набора 2104 примерных команд превращения, показывающих его использование. Таблица 2102 перечисляет название превращения и приемлемые типы операции для каждого превращения. Любой заголовок столбца с концевым "=" указывает то, что значение, представленное ниже его, может представляться в формате ключ/значение в конструировании синтаксиса команд. Столбцы "Ввод" и "Вывод" могут указывать ожидаемые типы/структуры данных для превращения/операции в контексте Python v3.6. Например, команда "serialize json sortkeys=t" может выполнять следующие последовательности манипулирования данными: принимать в качестве ввода любую Python-структуру данных, выполнять json.dumps для нее с флагом "sort keys", заданным как истина, а затем выводить Python-строку с преобразованной в последовательную форму версией данных. Обратный режим этой команды может ожидать JSON-форматированной Python-строки в качестве ввода, выполнять json.loads для нее, а затем выводить Python-структуру данных. Флаг "sort_keys" информирует функцию json.dumps в отношении необходимости обрабатывать ключи любой словарной Python-структуры в порядке возрастания. Python v3.6 не может гарантировать согласованный порядок обработки для структуры словаря при обработке посредством ключей, в силу чего результирующие JSON-строки могут быть несогласованными между несколькими сериями этого превращения в идентичной структуре данных. Сортировка ключей в конкретном порядке в превращении с преобразованием в последовательную форму может предоставлять согласованность в последовательности обработки, что приводит к идентичным JSON-

строкам в качестве вывода между несколькими сериями в идентичной структуре данных. Это может становиться очень важным для целей определения того, являются или нет две JSON-строки эквивалентными, и по сути может представлять две эквивалентных структуры данных до преобразования в последовательную форму.

Превращение со сжатием в табл. 2102 показывает несколько различных операций сжатия без потерь или обратимых сжатий. Любые операции необратимого сжатия или сжатия с потерями могут расширять репертуар превращения со сжатием, но для целей пояснения обратимых превращений, может быть неинтересным и неконструктивным пояснять одностороннюю функцию, которая может не предоставлять криптографическую цель сильно за рамками уменьшения размера данных. С точки зрения TOP, сжатия с потерями могут анализироваться и трактоваться идентично превращению дайджеста, которое поясняется в последующем разделе. В примере на 2104, команда "compress bz2" может выполнять bz2-сжатие для ввода двоичной строки и может формировать вывод двоичной строки, который может иметь или не может иметь меньший размер, чем входная строка. Некоторые данные более не могут быть сжимаемыми с использованием конкретной схемы сжатия; пример означенного может представлять собой ситуацию, когда bz2-сжатая строка может обрабатываться снова, и дополнительное уменьшение размера данных не может достигаться.

Превращение с кодированием

Фиг. 22 показывает таблицу спецификаций команд для превращения 2202 с кодированием и набора 2204 примерных команд превращения, показывающих его использование. Может быть предусмотрено множество схем кодирования в компьютерной науке, и ссылки в этой таблице не представляют все известные схемы кодирования. Схемы кодирования, которые перечислены в столбце "кодирование=", могут содержаться в Python v3.6 и его ассоциированных стандартных библиотеках. Специалисты в данной области техники могут распознавать полезность наличия доступа ко всем этим типам кодирований для решения проблемы, связанной с приложением, которое может манипулировать данными. "Кодеки (98)" означают список поддерживаемых кодеков в Python v3.6 на момент написания данного материала и перечислены ранее в таблице на фиг. 11. Команда превращения "encode strbin utf 8" может принимать в качестве ввода Python-строку, выполнять utf_8 Unicode-кодирование для нее и выводить результаты в качестве байтовой Python-строки. Команда превращения "encode utf utf_16" может принимать в качестве ввода Python-строку, выполнять utf 16 Unicode-кодирование для нее и выводить результаты в качестве Python-строки. Команда превращения "encode binascii hex" может принимать в качестве ввода байтовую Python-строку, выполнять шестнадцатеричное кодирование для нее и выводить результаты в качестве Python-строки. Команда превращения "encode base 64" может принимать в качестве ввода байтовую Python-строку, выполнять base64 двоичное кодирование для нее и выводить результаты в качестве Python-строки. Команда превращения "encode utf_8" является эквивалентной "encode utf utf_8". Эти пояснения могут иллюстрировать согласованность и типы перестановок, разрешенных в синтаксисе команд превращения с кодированием.

Превращение дайджеста

Фиг. 23 показывает таблицу спецификаций команд для превращения 2302 дайджеста и набора 2304 примерных команд превращения, показывающих его использование. Превращение дайджеста, как показано в таблице 2302, задает три типа операций, но не только: hash, hmac и/или cmac. Команда превращения "digest hash md5 128" может принимать в качестве ввода строку исходных данных, выполнять хеш-функцию MD5 для нее и формировать выходную байтовую строку дайджеста, которая имеет длину в 128 битов. Следует отметить, что входная исходная строка данных не может модифицироваться и не может перезаписываться во время вызова превращения дайджеста; выходная байтовая строка дайджеста может представлять собой дополнительные данные, сформированные из вызова превращения дайджеста, и может предоставляться отдельное пространство в запоминающем устройстве. Команда превращения "digest hash sha2 512" может принимать в качестве ввода строку исходных данных, выполнять SHA2-хеш-функцию для нее и формировать выходную байтовую строку дайджеста, которая имеет длину в 512 битов. Команда превращения "digest hash shake256 digestlen=332" может принимать в качестве ввода строку исходных данных, выполнять хеш-функцию SHAKE256 для нее и формировать выходную байтовую строку дайджеста, которая имеет длину в 332 бита. Команда превращения "digest hmac sha2 256" может принимать в качестве ввода строку исходных данных, выполнять HMAC-функцию для нее с использованием SHA2-хеша и формировать выходную байтовую строку дайджеста, которая имеет длину в 256 битов. Команда превращения "digest cmac aes 256" может принимать в качестве ввода строку исходных данных и 256-битовый симметричный ключ, выполнять CMAC-функцию для них с использованием AES256-шифра и формировать выходную байтовую строку дайджеста, которая имеет длину в 128 битов. Все эти примерные операции и типы превращения дайджеста могут содержаться в стандартной Python-библиотеке и/или PyCryptodome-библиотеке и могут не представлять все множество операций, типов, длин дайджестов, длин ключей и/или других параметров, которые могут существовать в теоретическом и/или реализованном смысле за пределами этих примерных библиотек. Любые дополнительные варьирования могут надлежащим образом анализироваться через TOP и интегрироваться в форму превращения. Такие интеграции для любой формы превращения могут требовать рефакторинга и повторного тестиро-

вания существующих операций превращения.

Превращения а-шифра/ц-подписи

Фиг. 24 показывает таблицу спецификаций команд для превращений 2402, 2404, 2406 а-шифра и ц-подписи и набора 2410 примерных команд превращения, показывающих его использование. Команда превращения "acipher шифр pkcsl_oaep 2048" может принимать в качестве ввода байтовую строку и асимметричный открытый RSA-ключ с длиной в 2048 битов, выполнять операцию OAEP-шифрования RSA PKCS#1 для них с использованием 512-битового SHA2-хеша и может формировать в качестве вывода зашифрованную байтовую строку, которая имеет длину в 2048 битов. Команда превращения "acipher pkcs1 v1 53072" может принимать в качестве ввода байтовую строку и асимметричный открытый RSA-ключ с длиной в 3072 бита, выполнять операцию шифрования RSA PKCS#1 v1.5 для них и может формировать в качестве вывода зашифрованную байтовую строку, которая имеет длину в 3072 бита. Обратный режим этих превращений а-шифра может требовать в качестве ввода зашифрованный текст в качестве байтовой строки и закрытую часть соответствующего RSA-ключа, чтобы формировать исходный открытый текст.

Команда превращения "dign pkcs1_v1_52048" может принимать в качестве ввода исходную байтовую строку и асимметричный закрытый RSA-ключ с длиной в 2048 битов, выполнять операцию применения цифровой подписи RSA PKCS#1 v1.5 для них с использованием 512-битового SHA2-xeina и может формировать в качестве вывода байтовую строку дайджеста, которая имеет длину в 2048 битов. Следует обратить внимание, что термин "байтовая строка дайджеста" может использоваться наравне с "байтовой строкой цифровой подписи", поскольку TOP может рассматривать эти выводы как предоставляющие аналогичную функциональность и в силу этого может сохранять такую байтовую строку, упоминаемую посредством имени переменной "digest". Команда превращения "dign dss 1024 hashtyp=sha2" может принимать в качестве ввода исходную байтовую строку и асимметричный закрытый DSA-ключ с длиной в 1024 бита, выполнять операцию применения цифровой DSS-подписи для них в FIPS-186-3-режиме с использованием 512-битового SHA2-хеша и может формировать в качестве вывода байтовую строку дайджеста, которая имеет длину в 1024 бита. Команда превращения "dign dss 256" может принимать в качестве ввода исходную байтовую строку и асимметричный закрытый ECC-ключ с длиной в 256 битов, выполнять операцию применения цифровой DSS-подписи для них в FIPS-186-3-режиме с использованием 512-битового SHA2-xeina и может формировать в качестве вывода байтовую строку дайджеста, которая имеет длину в 256 битов. Обратный режим этих превращений ц-подписи может требовать в качестве ввода байтовую строку дайджеста (цифровую подпись), исходную байтовую строку и открытую часть соответствующего асимметричного ключа, чтобы аутентифицировать его.

Превращение с извлечением

Фиг. 25 показывает таблицу спецификаций команд для превращения 2502, 2504, 2506 с извлечением и набора 2510 примерных команд превращения, показывающих его использование. Примерные операции pbkdf2, hkdf и scrypt также могут быть известными как функции извлечения ключа и/или функции растягивания ключа. Базовая функциональность превращения с извлечением может заключаться в том, чтобы извлекать симметричный ключ или ключи требуемой длины из строки двоичных или символьных данных, которая может быть известна для пользователя; широкое использование функции извлечения ключа может заключаться в том, чтобы извлекать надлежащим образом сформированный симметричный криптографический ключ(и) из пароля или фразового пароля. Команда превращения "derive pbkdf2 keylen=256 iterations=100000" может принимать в качестве ввода строку символьных данных (пароль или фразовый пароль), выполнять PBKDF2-операцию для нее с использованием 512-битовой SHA2-хеш-функции, случайно сформированного 512-битового вектора инициализации в качестве соли и параметра подсчета итераций, заданного равным 100000, и может формировать соответствующий симметричный ключ, который представляет собой байтовую строку данных с длиной в 256 битов. Команда превращения "derive hkdf keylen=256 numkeys=4" может принимать в качестве ввода байтовую строку данных, выполнять HKDF-операцию для нее с использованием 512-битовой SHA2-хеш-функции, случайно сформированного 512-битового вектора инициализации в качестве соли и может формировать соответствующий набор из четырех связанных симметричных ключей, каждый из которых представляет собой байтовую строку данных с длиной в 256 битов. Команда превращения "derive scrypt keylen=128 mode=login" может принимать в качестве ввода строку данных, выполнять SCRYPT-операцию в режиме входа в учетную запись для нее с использованием случайно сформированного 512-битового вектора инициализации в качестве соли и может формировать соответствующий симметричный ключ, который может представлять собой байтовую строку данных с длиной в 256 битов. Режим входа в учетную запись превращения с извлечением сценария может быть сокращением для задания трех параметров n, r и p равным значениям, указываемым в табл. 2506. Эти значения параметров могут представлять собой предлагаемые настройки автора SCRYPT-алгоритма.

TOP-подход к превращениям с извлечением может предлагать бимодальный режим работы. Режим данных: если превращение может участвовать без стека ключей (который поясняется подробно в последующем разделе), а только с исходной строкой данных некоторого типа, оно может превращать эту входную исходную строку данных и заменять ее выводом превращения, которое может иметь форму

симметричного ключа(ей). Режим ключей: если превращение может участвовать со стеком ключей и источником данных некоторого типа, оно может превращать соответствующий исходный материал ключа, присутствующий в стеке ключей, и может заменять исходный материал ключа, за счет этого извлекая криптографически применимый симметричный ключ(и) и помещая его в стек ключей. Эти операторы могут подробнее проясняться в последующем разделе, когда стеки ключей и управление ключами поясняются в контексте записи с результатами аудита превращений или TAR и зависимых превращений.

Превращение s-шифра

При использовании TOP, операции симметричного шифрования могут классифицироваться в качестве превращений s-шифра и в качестве группы, эти превращения могут представлять набор ассоциированных атрибутов, которые могут быть обширными по числу и разнообразию. Следующие три чертежа иллюстрируют то, как TOP может систематически нормализовать и инкапсулировать каждое превращение s-шифра со всеми своими атрибутами в идентичную выходную строку. Этот тип технологий встраивания атрибутов может содержаться в различных функциях и библиотеках для множества типов операций. Тем не менее, имеется очень небольшое число общепризнанных стандартов для таких технологий встраивания. TOP может предлагать согласованную технологию, которая может применяться ко всем превращениям s-шифра в различных целях поддержки признака, называемого складыванием структурированных данных с превращениями или SDFT. То, может или нет такая технология становиться широко используемым стандартом, может выходить за рамки этого документа, но читатель может распознавать возможные преимущества ее использования в TOP-инфраструктуре, в частности, когда ниже поясняются TAR- и SDFT-конструкты и способы.

Фиг. 26 показывает таблицу спецификаций команд для превращения 2602 s-шифра и набора 2604 примерных команд превращения, показывающих его использование. Таблица показывает три типа операций s-шифрования: aes, chacha20 и salsa20. Он не представляет собой полный список известных симметричных шифров, но он может представлять их релевантное множество, чтобы иллюстрировать то, как TOP может организовывать их и предлагать их использование. Симметричные шифры могут иметь следующие атрибуты, ассоциированные с ними в большей или меньшей степени: длина ключа (дл. ключа), рабочий режим (режим), тип соли (тип соли), длина соли (дл. соли), размер блока (блок), типы операций шифрования (тип) и/или дополнение (доп.). Длина ключа может указывать длину секретного симметричного ключа, который может использоваться в шифре, чтобы формировать зашифрованный текст из открытого текста. Для AES-шифров, они могут иметь по меньшей мере десять различных рабочих режимов, как показано в таблице. Большинство симметричных шифров может требовать ввода соли (случайного значения) конкретного типа (вект.ин или одноразовый номер) и конкретную длину соли, использование которой может стимулировать лучшую семантическую безопасность. Симметричный шифр может предоставлять по меньшей мере три различных типа операций: блочный, потоковый и/или AEAD-шифры. Более новые режимы могут предлагаться, и они могут интегрироваться с использованием TOP в качестве дополнительной разновидности превращения. Шифры блочного режима могут требовать дополнительных атрибутов, содержащих технологию дополнения, позиционирование дополнения, тип дополнения и/или длину заполнения.

В примерах в секции 2604, команда превращения "scipher aes 256 mode=ofb" может принимать в качестве вводов байтовую строку данных и 256-битовый симметричный ключ, шифровать входную строку данных с использованием потокового шифра на основе AES-256 OFB-режима с представленным ключом и случайно сформированным 128-битовым вектором инициализации и формировать выходную строку, которая может состоять из зашифрованного текста и всех ассоциированных атрибутов, предусмотренных в процессе, встроенном в заголовок выходной байтовой строки, отформатированной в согласованном формате ключ/значение, как указано на фиг. 27 (который поясняется в последующем разделе). Команда превращения "scipher aes 128 mode=gcm" может принимать в качестве вводов байтовую строку данных и 128-битовый симметричный ключ, шифровать входную строку с использованием потокового AEAD-шифра на основе AES-256 GCM-режима с представленным ключом и 128-битовым одноразовым номером и формировать выходную байтовую строку, которая может состоять из зашифрованного текста и всех ассоциированных атрибутов, предусмотренных в процессе, встроенном в заголовок выходной строки, отформатированной в согласованном формате ключ/значение, как указано на фиг. 27. AEAD является аббревиатурой для аутентифицированного шифрования с ассоциированными данными и может быть стандартизированным или известным способом встраивания функциональности аутентификации наряду с возможностями шифрования симметричного шифра в пределах одного вызова функции. Команда превращения "s-шифр chacha20256" может принимать в качестве вводов байтовую строку данных и 256-битовый симметричный ключ, шифровать входную строку с использованием потокового CHACHA20-шифра с представленным ключом и 64-битовым одноразовым номером и формировать выходную строку, которая может состоять из зашифрованного текста и всех ассоциированных атрибутов, предусмотренных в процессе, встроенном в заголовок выходной строки, отформатированной в согласованном формате ключ/значение, как указано на фиг. 27. Команда превращения "s-шифр salsa20 128" может принимать в качестве вводов байтовую строку данных и 128-битовый симметричный ключ, шифровать входную строку с использованием потокового SALSA20-шифра с представленным ключом и 64-битовым однора-

зовым номером и формировать выходную строку, которая может состоять из зашифрованного текста и всех ассоциированных атрибутов, предусмотренных в процессе, встроенном в заголовок выходной байтовой строки, отформатированной в согласованном формате ключ/значение, как указано на фиг. 27.

Фиг. 27 показывает формат структуры вывода для выходной строки s-шифра в последовательности из двух этапов, при этом этап 1 иллюстрирует формат ввода, а этап 2 иллюстрирует формат вывода. "Заголовок" представляет собой строку utf8-кодированных параметров ключей/значений переменной длины превращения s-шифра для выходного сообщения. На этапе 1, s-шифр может подтверждать в качестве ввода байтовую строку сообщения "сообщение" переменной длины с необязательным дополнением длины заполнения, обычно размещаемым в конце сообщения по мере необходимости. К началу сообщения может добавляться значение соли, которое может рекомендоваться посредством выбранного шифра. Дополнение может представлять собой обязательное требование симметричного шифра блочного режима. Если конкретная технология дополнения не указывается программистом, технология дополнения по умолчанию может использоваться и добавляться в конец сообщения. Это сообщение и дополнение могут упоминаться как простой текст. Выбранный шифр теперь может обрабатывать входной простой текст и формировать вывод, который может называться зашифрованным сообщением, как показано на этапе 2. Выбранное превращение s-шифра теперь может подготавливать встроенный заголовок в качестве печатаемых пар ключ/значение в формате строки символов, в котором ключи могут представлять тип параметра, и значения представляют их соответствующие настройки. В следующем разделе поясняются подробности ключа/значения. После того, как строка заголовка может формироваться, превращение может вычислять длину этой строки заголовка, называемой в качестве размера заголовка, и может форматироваться в качестве целого числа с обратным порядком байтов без знака с длиной в два байта. Два байта могут варьироваться по значению от 0 до 2^{16} (65536) и могут быть достаточными, чтобы описывать все атрибуты для любых симметричных шифров для обозримого будущего в этом конкретном формате. Затем этап 2 может продолжаться, чтобы создавать пакетированное сообщение, содержащее размер заголовка, заголовок и зашифрованное сообщение. Это пакетированное сообщение может представлять собой фактическую выходную строку из превращения s-шифра, в силу чего она может рассматриваться как имеющая успешно инкапсулированные и встроенные все атрибуты, ассоциированные с превращенными данными. Потоки данных обратного превращения s-шифра могут придерживаться этого процесса наоборот: команда превращения может указывать точное превращение s-шифра, которое следует выполнять, совпадающий симметричный ключ и пакетированное сообщение могут предоставляться в качестве вводов. Превращение s-шифра затем может распаковывать пакетированное сообщение, читать и сохранять все атрибуты, содержащиеся в заголовке, после чего подготавливаться к расшифровке зашифрованного сообщения. Симметричный шифр может не иметь детерминированного способа для того, чтобы передавать успешное дешифрование. Способ верификации может использоваться наложенным способом для того, чтобы определять такие результаты. Пример рудиментарного способа может заключаться в том, чтобы извлекать добавленное к началу значение соли из дешифрованного сообщения и сравнивать его с сохраненным значением соли из заголовка. Совпадающие значения соли могут указывать успешное дешифрование, но не могут гарантировать его. Симметричные шифры в AEAD-режиме могут разрешать эту проблему в большей степени посредством встраивания MAC (CMAC, HASH или HMAC) строки данных (до или после шифрования) в зашифрованном сообщении и выполнения сравнения. Более сложные способы могут требовать аутентификации хешей с цифровой подписью некоторой формы сообщения с использованием различных ключей. Как можно видеть в последующем разделе, использование SDFT и TAR может обеспечивать такой уровень сложности процедурно простым и логическим способом. Во всех этих хеш-технологиях, может быть детерминированно невозможным полностью утверждать условие попытки дешифрования вследствие слабых мест, внутренне имеющихся в схеме хеширования, чтобы уникально идентифицировать данные универсально. Один детерминированный способ может заключаться в том, чтобы сравнивать дешифрованное сообщение с исходным сообщением на предмет равенства, но могут быть компромиссы эффективности для длинных сообщений.

Фиг. 28 показывает таблицу ключевых слов параметров и технических требований для строки заголовка в формате структуры вывода превращения s-шифра. Ключевые слова, выбранные для этой таблицы атрибутов, могут быть в достаточной степени самоописываемыми и/или самоочевидными для специалистов в данной области техники. Примеры значений атрибутов показаны в столбце справа. Первый перечисленный атрибут, размер заголовка, может представлять собой единственный атрибут, который может представляться в качестве 16-битового двоичного целочисленного значения с обратным порядком байтов без знака, и может быть первым полем, присутствующим в заголовке. Этот размер заголовка может указывать число байтов, которые следуют далее, которое может описывать атрибуты этого конкретного превращения s-шифра в печатаемом формате. Формат атрибута может выбираться как печатаемый, чтобы обеспечивать вариативность диапазонов и длин значений атрибутов. Все значения атрибутов, которые могут существовать естественно в качестве двоичной строки в выполняющейся программе, содержащей значения соли (salt val) и MAC-строки (mac val), могут кодироваться в base64, чтобы удовлетворять предпочтению печатаемых символов.

Таким образом, выходная строка превращения s-шифра может содержать один или более инкапсу-

лирующих уровней атрибутов в зависимости от подробностей выбранного s-шифра. Фиг. 29 показывает иллюстрацию итеративных встроенных инкапсуляции сообщений для превращения s-шифра в AEAD-режиме. AES-шифр AEAD-режима может выводить следующие уровни, перечисленные от внутренних до внешних уровней. Уровень 2910 подготовки сообщений может содержать сообщение открытого текста, которое должно шифроваться 2914, комбинированное с соответствующим значением 2912 соли. Это подготовленное сообщение 2910 может шифроваться с помощью выбранного AEAD-шифра, который затем может дополнительно формировать MAC-значение и дополнительные данные 2922 в качестве части процесса шифра, и можно обратиться к этому комбинированному сообщению в качестве вывода 2920 AEAD-шифра. Этот вывод 2920 AEAD-шифра также может упоминаться как зашифрованное сообщение 2934. Зашифрованное сообщение 2934 может иметь ассоциированные атрибуты из процесса s-шифрования, которые могут параметризоваться с использованием способа заголовка ключевого слова/значения из фиг. 28, чтобы формировать заголовок 2932, и эта комбинация может упоминаться как пакетированное сообщение 2930 s-шифра. Это пакетированное сообщение 2930 s-шифра может представлять собой вывод выбранного превращения s-шифра, которое может сохраняться в указателе или переменной 2944 объекта NSstr-структуры 2940, которая может быть ассоциирована с TAR, которая вызывает превращение s-шифра. В последующем разделе подробнее поясняется структура NSstr. Также другие атрибуты 2942 могут сохраняться в этой единице хранения данных, называемой NSstr-структурой, содержащей флаги TAR, стека ключей, дайджеста и/или состояния. Указатель или переменная 2944 объекта в NSstr-структуре 2940 может представлять собой начальную точку сообщения 2914 открытого текста, в силу чего итеративный тракт 2950 может быть возможным и может существовать для объекта 2944 для стольких вложенных инкапсуляции, сколько требуется посредством TAR, которую она может обрабатывать, непосредственно которая может сохраняться в атрибутах 2942 NSstr-структуры 2940.

В заголовке 2932 пакетированного сообщения 2930 s-шифра, параметры, содержащие описание симметричного шифра, его используемые значения режима и атрибутов могут полностью и точно описываться посредством ключевых слов, перечисленных на фиг. 28. В этом отношении, TOP-подход может основываться не на запутывании и скрытии некриптографических процессов и процедур для защиты данных, а вместо этого только на теоретической и реализованной безопасности шифра, используемого в качестве превращения. Это может не казаться значительным при начальном наблюдении, но ниже можно показывать, что такая четкость ассоциированных подробностей трансформаций данных, встроенных в вывод самой трансформации, может в конечном счете быть подходящей для новых технологий и проектных решений, которые могут базироваться больше на самоописываемых данных, чем аппаратно-реализованные программы, чтобы надлежащим образом обрабатывать их. Этот подход может помогать формулировать один из фундаментальных примитивов в датацентрических проектных решениях и датацентрических моделях управления данными на некоторых самых нижних уровнях науки хранения данных. NUTS может базироваться в большой степени на датацентрических проектных решениях и моделях, как можно видеть в последующем разделе.

Фиг. 30 показывает таблицу спецификаций команд для превращения 3002 замков и набора 3010 примерных команд превращения, показывающих его использование. Превращение замков представляет собой одно из дополнительных превращений, перечисленных в таблице на фиг. 12, и оно может представлять собой вариант осуществления изменяемого замка из NUTS, как подробно описано на фиг. 65-77. Изменяемые замки могут представлять несколько различных способов криптографического замыкания секретного сообщения, содержащего SS-замок, OR-замок, MAT-замок, XOR-замок и хеш-замок. Признак изменяемых замков может представлять собой способность вводить и использовать несколько криптографических ключей в процессе шифрования секретного сообщения нормализованным, систематическим и упорядоченным способом. TOP-подход может обеспечивать возможность компактного способа реализации таких технологий замыкания простым и удобным способом. В примерах секции 3010, команда превращения "lock orlock 128 numkeys=10 scipherkeylen=128" может принимать в качестве вводов байтовую строку данных и до десяти 128-битовых идентичных симметричных ключей, шифровать входную строку с использованием команды превращения "scipher aes 128 mode=eax" и формировать выходную строку, содержащую зашифрованный текст и ассоциированные встроенные атрибуты. Команда превращения "lock matlock 256 numkeys=5" может принимать в качестве вводов байтовую строку данных и пять 256-битовых симметричных ключей, итеративно шифровать входную строку с использованием команды превращения "scipher aes 256 mode=eax" для каждого ключа и формировать выходную строку, содержащую зашифрованный текст и ассоциированные встроенные атрибуты. Команда превращения "lock sslock 256 numkeys=4 threshold=2" может принимать в качестве вводов байтовую строку данных и по меньшей мере два 256-битовых ключа по принципу разделения секретов tines256, шифровать входную строку с использованием реализации способа разделения секретов по принципу Шамира с предоставляемыми квотами секретов и формировать выходную строку, содержащую зашифрованный текст и ассоциированные встроенные атрибуты. Команда превращения "lock sslock_b 256 numkeys=6 threshold=3" может принимать в качестве вводов байтовую строку данных и по меньшей мере три 256-битовых ключа по принципу разделения секретов tinesidx128, шифровать входную строку с использованием реализации способа

разделения секретов по принципу Шамира с предоставляемыми квотами секретов и формировать выходную строку, содержащую зашифрованный текст и ассоциированные встроенные атрибуты. Команда превращения "lock xorlock 128 numkeys=6" может принимать в качестве вводов байтовую строку данных и шесть 128-битовых симметричных ключей, извлекать вычисленный ключ посредством итеративного выполнения XOR-операций для предоставляемых ключей, шифровать входную строку с использованием команды превращения "scipher aes 128 mode=eax" и формировать выходную строку, содержащую зашифрованный текст и ассоциированные встроенные атрибуты. Команда превращения "lock hashlock 192 numkeys=7" может принимать в качестве вводов байтовую строку данных и семь симметричных ключей на 192 бита, извлекать вычисленный ключ посредством выполнения хеша для упорядоченной конкатенации предоставляемых ключей, шифровать входную строку с использованием команды превращения "scipher aes 192 mode=eax" и формировать выходную строку, содержащую зашифрованный текст и ассоциированные встроенные атрибуты.

Каждое описание типов изменяемых замков и режима работы могут содержаться в последующих разделах относительно изменяемых замков, начиная с фиг. 60. TOP-анализ и способы могут предоставлять возможность осуществления сложных итеративных варьирований замыкания потенциально с использованием множества криптографических ключей точным логическим способом и могут обеспечивать легко достижимые расширения различных типов алгоритмов замыкания в будущем. Так же, ниже можно показывать, что аспект управления ключами SDFT может обеспечивать возможность программисту легко формировать и управлять таким множеством криптографических ключей с относительной простотой.

Как представлено на фиг. 12, 13 и 14, TOP-анализ и способы могут обеспечивать возможность специалистам в данной области техники принимать означенную функцию манипулирования данными и определять ее пригодность для нормализации в операцию и тип превращения. Таблица на фиг. 12 может показывать дискретизацию очень хорошо известного манипулирования данными и может считаться достаточной для использования посредством широкой аудитории разработчиков. Тем не менее, в таких случаях, когда функция манипулирования данными не может содержаться в этой таблице, может быть возможным анализировать и индивидуально адаптировать функцию с возможность работать в рамках SDFT-инфраструктуры с использованием TOP-способов; таких функций, как, но не только, сжатие с потерями, рассеяние битов, рассредоточения сообщений, кодирование со стиранием ошибок (ECC) и уровень сообщения кодирование RAID и структурирование. В большинстве случаев таких расширений превращения, может быть необязательным перекодировать или перезаписывать фактическую функцию манипулирования данными. Фактически, осуществление этого может быть контрпродуктивным и процедурно слабым в большинстве обстоятельств. К библиотеке, содержащей функцию манипулирования данными, может осуществляться доступ посредством библиотеки преобразований, и TOP-способ может обеспечивать возможность разработчику предоставлять нормализующую функцию-обертку вокруг конкретной функции манипулирования данными, чтобы демонстрировать хорошее поведение в рамках SDFT-инфраструктуры.

Структуры превращений записи с результатами аудита превращений (TAR) и складывание структурированных данных с превращениями (SDFT).

Фиг. 31 показывает технические требования различных структур превращений в табличном формате. Структурное определение основывается на вложенном подходе на основе ключей, аналогичном тому, как структуры задаются в рамках JavaScript; он может представлять собой намеренное проектное решение таким образом, что его представление может легко дублироваться в широком спектре языков программирования и может не основываться на особенностях конкретного языка. Например, Python v3.6 предоставляет возможность задания классов, тогда как некоторые языки, такие как JavaScript, не предоставляют, в силу чего структуры данных превращений не могут основываться на классах, чтобы задавать их для более широкой применимости. Структура превращений может предоставлять четко определенную область оперативного запоминающего устройства, за счет чего вводы и выводы превращения могут подготавливаться, обработанные и/или сохраненные. Основная единица или структура хранения данных, которая может использоваться в большинстве, если не во всех операциях превращения, называется NSstr-структурой 3108. Может быть предусмотрен по меньшей мере один экземпляр NSstr, ассоциированной с вызовом превращения. Все структуры превращений могут иметь поле typ или типа структуры, указывающее то, какую структуру оно представляет. NSstr-структура дополнительно может задавать поле state, указывающее состояние структуры и/или ее данных. Поле obj или объекта либо может хранить одно значение, либо оно может представлять собой указатель, который ссылается на другую область запоминающего устройства. Поле obj может представлять собой поле, в котором могут содержаться входные данные в большинство превращений. Также поле obj может представлять собой поле, в котором могут содержаться выходные данные большинства превращений. Поле digest, если оно существует, может сохранять дайджест данных, сохраненных или указываемых ссылкой посредством поля obj. Способ, которым может формироваться дайджест, может зависеть от конкретной ц-подписи или превращения дайджеста и предоставляемых параметров и/или атрибутов в команду превращения. "Стек ключей", если он существует, может представлять собой один экземпляр структуры KISS (структуры спецификации обме-

на ключами, которая поясняется в последующем разделе), либо он может представлять собой список KISS-структур в предварительно определенном порядке, который соответствует его рабочей TAR. Стек ключей по существу хранит секретный ключ(и) различных типов, которые могут требоваться посредством определенных превращений. Поле "TAR" может указывать на экземпляр NStar-структуры 3106.

NStar-структура 3106 может указывать конкретные записи с результатами аудита превращений (TAR), которые могут применяться к входным данным, сохраненным в поле obj NSstr-структуры. TAR может представлять собой совокупность команд превращения в логическом порядке, которые могут целесообразно упорядочиваться, чтобы обрабатывать данные в NSstr упорядоченным способом с хорошим поведением, чтобы формировать одно "складывание" NSstr-данных. Этот процесс выполнения TAR на NSstr-структуре данных может упоминаться в качестве вызова функции "запутывания". С другой стороны, вызов функции "распутывания" может "раскладывать" фрагмент сложенных данных в NSstr-структуре с использованием идентичной TAR, основывающейся на внутренне присущих характеристиках обратимых превращений. Следовательно, обратимость превращений может становиться центральным признаком в складывании структурированных данных с превращениями (SDFT). SDFT-технология может использовать TAR в NSstr-структурах, чтобы итеративно превращать объект в во многом аналогично конвейерной операции для данных. Поскольку анализ может осуществляться для обратимого поведения каждой команды превращения в TAR, в силу этого любая TAR может вызываться в обратном режиме или в вызове функции распутывания. Эта тема может поясняться более глубоко, поскольку в следующих разделах могут представляться дополнительные необходимые ингредиенты, которые могут обеспечивать возможность таких операций.

NSbin-структура 3102 может выполнять конкретную функцию, которая может быть или может не быть релевантной только для Python v3.6. В Python v3.6, различие может проводиться способом, которым строка данных может сохраняться внутренне. Она может сохраняться в качестве "байтовой" строки или строки символов. Байтовая строка типа данных может указывать то, что информация, содержимая в переменной, может представлять собой последовательность двоичных байтовых данных. Строка символов может указывать то, что информация, содержащаяся в переменной, может представлять собой последовательность символов представления битов, кодированных в некотором типе схемы кодирования. Python v3.6 может использовать сложную внутреннюю схему управления, чтобы лучше всего определять то, как сохранять конкретную строку символов, поскольку различные кодирования могут требовать различных требований по хранению в расчете на "символ". Пример может представлять собой то, что UTF-8 может использовать кодовые единицы с длиной в 8 битов для того, чтобы представлять каждый символ, тогда как UTF-16 может использовать кодовые единицы с длиной в 16 битов для того, чтобы представлять каждый символ; эти варьирования могут требоваться для того, чтобы передавать различные международные наборы символов, при этом число символов на языке может очень отличаться от английского алфавита и в силу этого не может вписываться в перестановки 8 битов данных. Предпочтительный внутренний способ преобразования в последовательную форму превращений, TAR и SDFT могут представлять собой JSON, и JSON может не иметь собственной поддержки для того, чтобы преобразовывать "байтовый" Python-тип данных в одно собственный тип данных. Если преобразование осуществлено, вызов JSON-функции может резко сбоить с некоторым индикатором того, что конкретный тип данных не может поддерживаться. NSbin-структура может специально проектироваться для этого типа ситуации и может подставляться вместо любых "байтовых" строк данных и в силу этого может обеспечивать JSON-совместимость Python-переменной. "Байтовая" строка может кодироваться в base64-строку символов и сохраняться в поле b64 NSbin-структуры. Строковая байтовая переменная теперь может задаваться с возможностью указывать на эту NSbin-структуру, перезаписывающую исходные байтовые данные. Они могут представлять эквивалентные данные, но они могут предоставляться в различных кодированиях и структурах. Тем не менее, конечный результат может заключаться в том, что NSbin-структура может быть полностью JSON-совместимой и теперь может безопасно преобразовываться в последовательную форму с использованием JSON-функций без ошибок вследствие несовместимых типов данных.

В TOP-подходе, эти "байтовые" данные в преобразовании и замене NSbin-структуры могут упоминаться как превращение с "прижатием" из фиг. 12 и 33. В Python v3.6, превращение с прижатием, перечисленное в таблице 3302, может принимать любую допустимую Python-структуру или переменную и итеративно превращать каждую байтовую строку в эквивалентную NSbin-структуру, которая может приводить к Python-структуре, не имеющей любых байтовых типов данных. Специалисты в данной области техники могут индивидуально настраивать соответствующее превращение с прижатием для языка, отличного от Python v3.6 и его вызова JSON-функции, чтобы удалять такие источники ошибок преобразования данных в последовательную форму. Обратный режим "прижатия" может упоминаться как "отжатие" и может отменять преобразование и замену итеративно таким образом, что может восстанавливаться структура данных, включающая в себя исходные типы данных.

NSjson-структура 3104 может выполнять особенно полезную функцию хранения только данных, которые могут быть полностью JSON-совместимыми. Быстрый взгляд на поля, заданные для NSstr 3108, может предупреждать в отношении потенциальной проблемы, если структура непосредственно отправлена для JSON-преобразования в последовательную форму вследствие поля дайджеста, потенциально

хранящего значение дайджеста исходного объекта в форме двоичной строки или байтовой строки данных в Python v3.6. Следует обратиться снова к фиг. 12 и повторно представлять превращение "Мебиуса" для этой конкретной проблемы. Следует отметить, что любое обоснованное определение превращения Мебиуса до этого момента в этом описании может не становиться полностью понятным для читателя вследствие переплетающегося характера превращений и TOP-подхода. Превращение Мебиуса на фиг. 32 может превращать данную структуру из одной формы в другую круговым способом, но с небольшим скручиванием, аналогично ленте Мебиуса. Превращение Мебиуса может представлять собой важный механизм реализации складывания структурированных данных с превращениями посредством систематического преобразования NSstr-структуры в JSON-преобразуемую в последовательную форму структуру, такую как NSjson; процесс преобразования может встраивать рабочую TAR для NSstr полностью наряду с превращенными данными, за счет этого насыщая результирующий допускающий хранение объект самоописываемой характеристикой. Превращение Мебиуса может представлять собой вариант осуществления, который выполняет сущность складывания структурированных данных в SDFT-библиотеке удобным способом. Разработчик может решать выполнять SDF вручную с использованием логической комбинации команд превращения, исключая команду Мебиуса, но команда Мебиуса добавляет по меньшей мере один дополнительный логический этап, который может требовать от разработчика выполнять этот этап за пределами SDFT-библиотеки: способность преобразовывать в последовательную форму NSstr-структуру данных, из которой он управляет, и в другую структуру, такую как NSjson. Превращение Мебиуса может представлять собой последнюю команду превращения в TAR. Вследствие своей функциональности, она может представлять собой единственное логическое место, в котором может быть размещено превращение Мебиуса. Когда превращение Мебиуса обрабатывается, оно может принимать NSstr-структуру, из/на которой оно может работать, и превращать ее в NSjson-структуру. TAR, встроенная в NSstr-структуру, более не может существовать в полезной или доступной форме, в силу чего превращение Мебиуса может представлять собой последнюю команду превращения данной TAR, которая должна обрабатываться. Просто, превращение Мебиуса может прижимать NSstr-структуру, JSON-преобразовывать ее в последовательную форму, затем сохранять ее в NSjson-структуре, которая может сохраняться, передаваться, JSON-преобразовываться в последовательную форму, складываться или любая другая операция допустимых данных, которая может выполняться для таких структур. Может быть предусмотрен обратный режим по отношении к превращению Мебиуса, но другой способ рассмотрения этого превращения может заключаться в том, чтобы утверждать, что оно представляет собой круговое превращение: независимо от прямого или обратного режима, оно выполняет конкретную трансформацию данных в зависимости от типа структуры входных данных. Табл. 3204 показывает NSx-структуру, для которой NSjson может представлять собой разновидность. Если в будущем возникает потребность в дополнительных структурах превращений, отличных от структур превращений, заданных на фиг. 31, и они, возможно, должны приспосабливаться в превращении Мебиуса, эта таблица иллюстрирует то, как превращение Мебиуса может вести себя для любой структуры превращений, отличной от NSstr. Это может не быть полностью очевидным без фактического программирования с помощью SDFT, но превращение Мебиуса может логически подразумевать, что может не быть возможной TAR-обработки из восстановленной NSjson-структуры, если превращение Мебиуса не может работать для нее, чтобы преобразовывать ее в его исходную NSstr-структуру, которая может хранить TAR, которая может складывать ее. Чтобы инициировать этот цикл вращения Мебиуса с NSjson-структурой, Мебиус (обращение) может запускаться "с ходу" с вызовом функции Мебиуса из SDFT-библиотеки, чтобы формировать NSstr-структуру, осуществлять доступ к встроенной TAR и обрабатывать встроенную TAR наоборот. Это дополнительно может подразумевать, что команда превращения Мебиуса в TAR, которая по определению должна представлять собой первую команду, которая должна обрабатываться в обратном режиме, может безопасно игнорироваться во время обработки, поскольку она, возможно, уже выполнена посредством вызова функции запуска "с ходу", в силу чего она не может выполнять функциональность Мебиус несколько раз во время таких обращений. В этом упорядочении, сбой при игнорировании превращения Мебиуса в обратном режиме может потенциально формировать бесконечное колебание вызовов Мебиуса, которые непрерывно преобразуют NSstr в NSjson и обратно. Это может казаться окольным путем выражения таких операций, но это может формировать довольно компактные двунаправленные TAR, которые могут систематически встраиваться в выходные превращенные данные, за счет этого насыщая самоописываемой характеристикой сложенные данные. Эта характеристика может быть новой в том, что на нее может воздействовать во многом аналогично интерпретируемым сценариям, но в прямом или обратном режимах, чтобы выполнять операции для данных согласованным восстанавливаемым способом на любом языке и/или в операционных системах, которые могут поддерживать реализацию SDFT-библиотеки.

Фиг. 33 показывает таблицу спецификаций команд для превращений 3302, 3304 с прижатием, с очисткой и ключа и набора 3310 примерных команд превращения, показывающих его использование. Превращение с очисткой может быть служебной функцией, которая удаляет переходные или временные поля из NSstr-структуры. Определенные превращения могут иметь потребность в дополнительных временных полях во время обработки и могут создавать дополнительные поля в NSstr-структуре, чтобы со-

хранять и осуществлять доступ к ним. Создание и использование таких переходных полей в NSstr может осуществляться вдумчивым способом после анализа их совместного существования в TOP-подходе и минимизации их помех надлежащему функционированию любых других превращений. Может отсутствовать режим обращения для превращения с очисткой вследствие своей функциональности, в силу чего оно может безопасно игнорироваться. Это значение обращения может учитываться при предложении нового переходного поля в NSstr-структуре, поле не может существовать в обработке режима обращения превращения, в силу чего превращение наоборот не может зависеть от его существования для своего функционирования надлежащим образом. Важная функция превращения с очисткой может заключаться в том, чтобы удалять внутреннюю копию полного стека ключей, используемого в обработке TAR; либо она может удалять только секретные ключи в стеке ключей и преобразовывать KISS-структуры в замочные скважины. Они могут представлять собой одни из наиболее критических превращений в SDFT TAR-обработке, поскольку сбой на предмет надлежащей очистки NSstr до ее подготовки для хранения может приводить к хранению всех без исключения криптографических ключей, которые могут использоваться в конкретной TAR-обработке, и она может непреднамеренно сохраняться в открытом тексте наряду со сложенными данными. Эта ситуация может раскрывать секретные ключи и компрометировать некоторые или все зашифрованные данные в сложенных данных; это может не быть намеченной целью шифрования данных.

В табл. 3304, превращение ключей показано с некоторыми его операциями. Это превращение может составлять часть функциональности управления ключами SDFT и может работать главным образом для поля keystack посредством обращения к полю TAR NSstr-структуры. Превращение с проверкой ключей может анализировать сохраненную TAR и может формировать список шаблонов ключей. Если стек ключей вводится, он может сравниваться с такими шаблонами ключей, чтобы определять то, предоставлены или нет корректные типы ключей в надлежащей последовательности во входном стеке ключей. Например, если TAR требует двух различных 256-битовых симметричных ключей для двух превращений ключей, которые могут требовать ключей, она может формировать два шаблона ключей "symmetric 256" в списке, обозначающих то, что TAR ожидает, что стек ключей должен содержать такие ключи, если он может присутствовать. Табл. 3504 перечисляет некоторые из различных типов ключей. Пустой стек ключей или частично заполненный входной стек ключей также может надлежащим образом обрабатываться. Когда стек ключей не может вводиться при том, что TAR требует некоторых ключей, в таком случае она может указывать превращение "с формированием ключей". SDFT может участвовать в режиме формирования ключей, за счет чего надлежащие типы ключей согласно извлеченным шаблонам ключей могут создаваться и составленный в стек ключей для отправки в рабочую NSstr-структуру до TAR-обработки для данных, сохраненных в поле obj. Частичный режим "формирования ключей" может активироваться, когда может вводиться частично заполненный стек ключей. Превращения с проверкой и формированием ключей могут совместно определять то, могут или нет частично предоставляемые ключи в стеке ключей иметь надлежащий тип и выполняться надлежащей последовательности. После этого они могут переходить к формированию надлежащих ключей для отсутствующих ключей. Этот процесс может упоминаться как сценарий "отсутствующих зубов" управления SDFT-стеками ключей.

Практически отсутствуют примеры TAR с командами превращения ключей, поскольку они могут считаться настолько фундаментальными для надлежащей работы SDFT-библиотеки в NSstr-структуре с использованием TAR, что они могут неявно выполняться по умолчанию в каждом вызове в операции запутывания/распутывания, а не заставлять программиста помещать их в каждую TAR. Может оказаться, что только наличие возможности обработки TAR, которая может требовать криптографический ключ, может быть достаточной причиной для того, чтобы неявно осуществлять проверку на предмет надлежащего управления стеками ключей согласованно, неявно и/или автоматически. Процесс TAR-обращения может обрабатывать стек ключей в надлежащем обратном порядке. Осложнения могут возникать вследствие особенностей превращения с извлечением в режиме стеков ключей, который поясняется в последующем разделе относительно того, как SDFT справляется с такими ситуациями, называемыми в качестве TAR-группировок для зависимых превращений.

Фиг. 34 показывает таблицу для структуры спецификации обмена ключами или KISS. Эта структура может иметь по меньшей мере два режима работы: ключ или замочная скважина. Атрибуты ключа могут указываться посредством некоторых или всех полей, заданных в таблице, и дополнительные поля могут добавляться, чтобы расширять структуру, чтобы поддерживать другие атрибуты ключей по мере необходимости. TOP-подход в криптографические операции может заключаться в том, чтобы заставлять вид каждого криптографического превращения требовать совпадающей замочной скважины, указывающей точный тип ключа, требуемого для того превращения. Атрибуты могут включать в себя, но не только, практически уникальный идентификатор для самого ключа, вопрос или подсказку для фразового пароля или пароля, описание ключа и т.д. Если значение ключа может присутствовать в KISS-структуре, оно может упоминаться просто как ключ. Если значение ключа может отсутствовать в KISS-структуре, оно может упоминаться как замочная скважина. Это может указываться посредством значения в поле "ima". Имя поля может представлять собой сжатие ключа/замочной скважины "I'm a" и может читаться так для понятности. Столбец, называемый "B", может указывать требуемые значения для создания неза-

полненной KISS-структуры и вставки в нее ключа в целях ее помещения во входной стек ключей NSstr-структуры. Столбец, называемый "Формир.", может указывать эти поля, которые могут создаваться и заполняться автоматически во время превращения с формированием ключей из SDFT-библиотеки. В SDFT-дискуссии, предусматривающей TAR, все ключевые ссылки могут быть синонимичны с KISS-структурами соответствующего типа. Очевидно, что стек ключей может тесно соответствовать характеристикам обрабатываемой TAR, и что этот способ накопления команд превращения и накопления необходимых криптографических ключей в конкретной форме и последовательности может предоставлять возможность итеративной обработки любых входных данных через бесконечное число варьирований превращений, параметрических разбросов превращений и/или последовательных складываний данных. В этот момент описания TOP, можно начинать понимать переплетающийся характер различных компонентов SDFT и то, что полная оценка какой-то конкретной части может не быть раскрыта полностью линейным способом.

Фиг. 35 показывает таблицу для KISS-режимов 3502 работы, матрицы 3504, показывающей преобразования типов ключей/формирования полей, и определений 3506 типов ключей. Табл. 3506 перечисляет несколько типов ключей, распознанных посредством SDFT, но она могут не быть ограничены означенными, поскольку новые типы ключей могут добавляться и интегрироваться по мере необходимости. По меньшей мере для трех типов ключей, возможно, требуется некоторое пояснение, поскольку они могут быть структурированы конкретно для SDFT-библиотеки с использованием известных базовых типов ключей. Тип ключа "symmetriclist" может представлять собой матрицу или список симметричных ключей и может сохраняться в качестве значения ключа в одной KISS-структуре. Этот тип ключа может поддерживать, но не только, такие превращения, как замок и извлечение. Превращения замков по принципу разделения секретов, называемые SS-замком и SS-замком_b, соответственно, могут представлять две различных реализации алгоритма разделения секретов по принципу Шамира. Превращение замков в виде SS-замка может ожидать квоты секретов в конкретном формате, содержащем внутренний числовой индекс и квоту ключей в ключе с длиной в 256 битов. Это может называться в SDFT-библиотеке типом ключа tines256. Превращение замков в виде SS-замка b может ожидать квоты секретов в конкретном формате, содержащем внутренний числовой индекс и квоту ключей в ключе с длиной в 256 битов. Это может называться в SDFT-библиотеке типом ключа tinesidx256.

Табл. 3502 представляет собой матрицу, показывающую то, какие характеристики могут применяться к KISS-структуре в двух режимах, в которых она может существовать: ключ (или превращение) или замочная скважина. В режиме превращений(ей) KISS-структура предположительно может сохранять фактический криптографический ключ, чтобы формировать некоторую версию зашифрованного текста, который может включать в себя дайджесты и/или ц-подписи с ключом. Следовательно, его хранение может использоваться информационно, но должно встраиваться дополнительно с использованием криптографических функций, чтобы постоянно сохранять его защищенным способом. В режиме замочных скважин, KISS-структура предположительно должна иметь достаточно подробностей для того, чтобы подтверждать соответствующий криптографический ключ в качестве своего значения, чтобы формировать некоторую версию зашифрованного текста, которая может включать в себя дайджесты, ц-подписи с ключом и/или извлеченные ключи. Следовательно, ее хранение может быть обязательным и, возможно, не должно дополнительно защищаться посредством технологии встраивания, поскольку они может не содержать значение ключа в качестве замочной скважины.

Табл. 3504 представляет собой матрицу, показывающую то, какие поля могут быть обязательными, релевантными, вводимыми и/или сформированными посредством типа ключа. После анализа таблицы, очевидно, что KISS-структура может хранить соли, связанные с различными криптографическими операциями. Это может казаться избыточным в свете пояснения относительно встроенных заголовков s-шифра, но это пояснение солей может не представлять все изображение по солям. Как показано на фиг. 37, сохраняемость атрибутов 3704, 3714, ассоциированных с превращением, может рассеиваться между несколькими областями 3732, 3734, 3736 и 3738 хранения данных. TOP-подход может показывать то, что соли могут встраиваться в определенные криптографические операции наряду с результирующими выходными данными, поскольку они не могут раскрывать дополнительную информацию относительно сформированного зашифрованного текста. Тем не менее, когда анализируются превращения с извлечением ключа, обработанные в режиме стеков ключей, можно обнаруживать то, что может быть удобным и логичным сохранять ассоциированное значение соли в KISS-структуре. Типичный способ использования функции извлечения ключа может заключаться в том, чтобы должен подтверждать фразовый пароль в качестве ввода, комбинировать его с некоторым значением соли и формировать надлежащим образом сформированный криптографический ключ, такой как, но не только, симметричный ключ. Использование соли в этом случае может быть предусмотрено для семантической безопасности. Следовательно, может быть полная возможность того, что каждая замочная скважина, которая может подтверждать идентичный фразовый пароль, может иметь различную соль, так что результирующие секретные криптографические ключи могут отличаться друг от друга по любой рациональной причине. Этот извлеченный ключ может использоваться временным способом и отбрасываться после использования, в силу этого оставляя только замочную скважину в качестве свидетельства своего существования.

Поскольку продукт извлечения ключа типично может не сохраняться постоянно, поскольку он может использоваться в качестве секретного ключа, может вызываться вопрос, где можно сохранять его? TOP может сохранять его в соответствующей замочной скважине и может предпочитать, что SDFT должно сохранять эту замочную скважину наряду со сложенными данными, в силу чего каждая замочная скважина, которая может подтверждать идентичный фразовый пароль, может иметь хранилище, подходящее для собственного экземпляра значения соли. Программист может сохранять замочные KISS-скважины внешне полностью отличающимся способом. Упрощенная схема превращений в верхней части фиг. 37, который является идентичным на фиг. 5, становится больше похожей на схему в нижней части фиг. 37, когда различные компоненты TOP и SDFT могут вводиться. Табл. 3720 обобщает размещение атрибутов.

Много описано выше относительно синтаксиса и множества команд превращения, проанализированных и доступных через TOP и SDFT, но как TAR фактически выглядит на практике? Фиг. 36 показывает структуру TAR и перечисляет несколько примеров TAR. Секция 3602 указывает общую структуру записи с результатами аудита превращений или TAR. Объявление "TAR label01" указывает название или метку TAR, задающейся чуть ниже его. Все TAR-команды придерживаются объявления TAR-метки, и пустая строка указывает конец текущего TAR-определения. Следовательно, множество TAR могут объявляться в одном текстовом файле. Секция TAR-определения может включать TAR-метки в строку непосредственно либо посредством команды превращения. Это может быть аналогичным макропризнакам компилятора языка программирования; она может использоваться в качестве удобной функции, чтобы комбинировать известные TAR-конструкты в новую TAR без необходимости фактически копировать определение в саму TAR. Команды превращения могут вставляться в конкретную последовательность, чтобы обрабатывать целевую NSstr-структуру требуемым способом. TAR "test a01" может просто прижимать Python-объект данных в эквивалентную структуру, не имеющую байтовых Python-типов данных; для других языков, она может выполнять или не может выполнять идентичные функции, поскольку "прижатие" может быть конкретным для языка и/или окружения. TAR "test a02" выполняет превращение с прижатием дважды последовательно. Второе превращение с прижатием может не осуществлять функциональные изменения данных. Оно показывает расширение TAR при работе. TAR "test_a07" может прижимать данные, преобразовывать их в последовательную форму в JSON-строку, затем преобразовывать их в байтовую двоичную строку с использованием utf 32-кодирования. TAR "test a17" показывает то, как может выглядеть завершающее превращение Мебиуса. TAR "test a20" прижимает данные, преобразует их в последовательную форму в JSON-строку, преобразует их в utf_8-кодированную двоичную строку, шифрует их с использованием chacha20 с 256-битовым симметричным ключом и затем преобразует результирующую двоичную строку зашифрованного текста в base64-кодированную строку символов. Симметричный ключ для превращения s-шифра может ожидаться в стеке ключей NSstr, который может содержать одну KISS-структуру, хранящую значение 256-битового симметричного ключа. Альтернатива может заключаться в том, что стек ключей может не предоставляться, и функция запутывания переходит к формированию допустимого стека ключей с надлежащим образом сформированным случайным 256-битовым симметричным ключом, использует его для того, чтобы выполнять превращение s-шифра, и обеспечивает возможность программисту осуществлять выборку копии стека ключей (и в силу этого ключа внутри) по завершению. TAR "test a42" показывает пример TAR-групп и зависимых превращений: она должна прижимать данные, преобразовывать в последовательную форму в JSON-строку, преобразовывать их в двоичную строку, кодированную в utf_8, извлекать 256-битовый симметричный ключ из фразового пароля, предоставляемого в стеке ключей, затем выполнять chacha20-шифрование для данных с использованием извлеченного симметричного ключа. Последние два превращения могут иметь постоянную зависимость, поскольку шифр основывается на извлеченном ключе; в силу этого эта зависимость может группироваться в TAR с начальными <тегами>, помеченными таким способом. В прямом режиме, может не быть очевидного влияния TAR-группировок внутри TAR-определения, за исключением подчеркивания таких зависимостей визуальным способом. Тем не менее, TAR-группы могут играть значительную роль, когда дело доходит до TAR-обращений. Когда TAR подготавливается для процесса TAR-обращения, TAR-группы могут поддерживаться незатронутыми в качестве единицы, и его составляющие не могут обращаться. Фиг. 41 и 42 иллюстрируют несколько примеров TAR-обращений. TAR "test_a64" может выполнять пять превращений s-шифра и превращение ц-подписи DSS. Эта TAR может ожидать стек ключей, заполненный шестью ключами различных типов и длин в конкретном порядке. В секции 3610, может быть проиллюстрировано упрощенное представление шаблона ключей, который может соответствовать TAR "test a64". Этот шаблон ключей может использоваться посредством неявных превращений с проверкой и/или формированием ключей, чтобы проверять достоверность любых входных стеков ключей и/или формировать допустимый стек ключей для надлежащей обработки TAR.

Фиг. 38 показывает блок-схемы SDFT-операций запутывания и распутывания (или обращения запутывания). Две центральных операции в SDFT могут представлять собой "запутывание" и ее инверсию, "распутывание". Операция запутывания может обрабатывать данную NSstr, которая может содержать некоторые или все следующие элементы: данные, TAR, стек ключей и/или другие атрибуты. Операция запутывания может "запутывать" или складывать 3810 исходные данные на 3802 согласно последова-

тельности превращений, перечисленных в TAR на 3802, и может в конечном счете формировать вывод в качестве компонента в NSstr-структуре 3804 или NSjson-структуре 3806. Операция распутывания может "распутывать" или раскладывать 3820 исходную структуру NSstr 3804 или NSjson 3806 согласно обратной последовательности превращений, перечисленных во встроенной TAR, и может в конечном счете формировать вывод в качестве NSstr-структуры 3802. Как показано, симметрия запутывания/распутывания может быть интересным аспектом этого проектного решения. Следует отметить согласованность терминологии и перспектив, которые могут использоваться для всего TOP. Операция запутывания наоборот может быть эквивалентной распутыванию. Этот принцип обратимости может не только упрощать анализ таких функций, но он может проникать через модульные способы организации, которые могут приводить к концептам высшего порядка, связанным с превращением данных.

Фиг. 39 показывает блок-схему последовательности операций способа SDFT-операции запутывания. С учетом NSx-структуры, вызов функции или метода запутывания может выполнять следующие операции для данных, содержащихся внутри, с использованием либо параметров, предоставляемых с вызовом, и/либо TAR, встроенной в NSx, причем в этом случае "x" означает любую структуру превращений. Аналогично превращениям с проверкой/формированием ключей, превращение Мебиуса может считаться настолько фундаментальным для этого алгоритма, что оно может неявно выполняться для любой структуры входных данных, если условия удовлетворяются 3906. Запутывание может надлежащим образом выполнять только свои базовые операции для NSstr-структуре, в силу чего если NSx-структура, которая не представляет собой NSstr, моно может выполнять попытку преобразования в NSstr-структуру 3918. Сбой при формировании допустимой NSstr 3924 может вызывать соответствующий код 3978 ошибки и резко завершать процесс 3984. Может быть предусмотрено по меньшей мере три различных способа, посредством которых данные могут запутываться или складываться: во-первых, в допустимой NSstr, может содержаться TAR, указывающая последовательности превращений, которые следует выполнять для данных в NSstr-структуре; во-вторых, название TAR-метки может пересылаться в вызов запутывания в качестве параметра, в силу этого указывая предпочтительный набор превращений, которые следует выполнять для данных в NSstr-структуре; в-третьих, индивидуально настраиваемый TAR-список может пересылаться в качестве параметра наряду с его именем в вызове запутывания, в силу этого указывая предпочтительный набор превращений, которые следует выполнять для данных в NSstr-структуре. Подготовка TAR 3912 может содержать расширение других ссылок в форме TAR-меток и/или их надлежащее упорядочение для режима обхода, который может быть прямым или обратным. Фиг. 41 и 42 иллюстрируют несколько примеров TAR-обращений. Затем превращение с проверкой ключей может эффективно выполняться для TAR и NSstr-структуры. Компонент превращения с проверкой ключей может заключаться в том, чтобы извлекать список шаблонов ключей посредством анализа TAR 3930. При использовании TAR, входной стек ключей (который может быть пустым или частично заполненным) и/или шаблоны ключей, процесс может составлять стек ключей для надлежащего обхода TAR 3936. Он может содержать формирование отсутствующих ключей корректного типа, упорядочение ключей в надлежащем порядке и/или проверку введенных ключей на предмет надлежащей структуры и типа. Все рассогласования входных типов ключей и соответствующих шаблонов извлеченного ключа могут формировать состояние 3942 ошибки, приводящее к вызыванию соответствующего кода 3978 ошибки, и резко завершать процесс 3984. Процесс теперь может итеративно выполняться для каждой команды превращения в TAR в надлежащей последовательности 3948 и выполнять указанное превращение 3954 для данных, содержащихся в NSstr. Любые ошибки, которые могут встречаться во время выполнения команд превращения 3960, могут вызывать соответствующий код 3978 ошибки и резко завершать процесс 3984. Когда конец TAR-последовательности достигается 3948 без ошибок, в таком случае операция запутывания может считаться успешной 3966, и процесс может выходить корректно 3972.

Фиг. 40 показывает блок-схему последовательности операций способа SDFT-операции распутывания. Вместо подробного указания процесса распутывания, можно иллюстрировать симметрию обратимости превращений посредством сравнения блок-схем последовательности операций способа на фиг. 39 и 40. Единственное различие между двумя блок-схемами последовательности операций способа может заключаться в этапах 3912 и 4012 TAR-подготовки. Поскольку каждое превращение может анализироваться и структурироваться с использованием TOP, чтобы работать с хорошим поведением двунаправленным способом, процесс распутывания, возможно, не должен быть существенно отличающимся от процесса запутывания за исключением того, как может представляться TAR. Он может реализовываться как идентичный код, но иметь небольшое отклонение, когда обратный флаг может указываться, и выполнять надлежащее обратное упорядочение TAR в случае обнаружения. Такой вызов в Python v3.6 может принимать форму "obj.ravel (..., reverse=True)". Симметрия может обеспечивать возможность фактическому реализованному коду быть намного меньшим и/или может представлять меньше возможностей для программных ошибок. Концептуальное преимущество может заключаться в понятности и простоте идеи при конструировании новых TAR в конкретных целях: программист может основываться на надлежащей TAR-последовательности, которая должна быть полностью обратимой в пределах своих ограничений, и он, возможно, не должен много думать в отношении этой части приложения. Преимущество может заключаться в том, что рабочая нагрузка программиста при создании конкретного набора превра-

щений данных может эффективно уменьшаться, по меньшей мере, наполовину, поскольку он более, возможно, не должен создавать обратный код такого манипулирования данными. Компоновка сложных шифровальных и отмыкающих механизмов может требовать огромного числа манипулирований с данными с использованием большого числа криптографических ключей. Способы на основе принципа организации превращений (TOP) позволяют помогать достигать более связанных и объединенных способов приближения к такой сложности дискретными, менее подверженными ошибкам способами; в силу этого, они могут предоставлять возможность, но не только, более согласованного, надежного, защищенного, портативного, понятного, всестороннего, гибкого, наращиваемого и/или сложного кода и/или данных.

Фиг. 43 показывает таблицу превращений, преобразованных в шаблон типов ключей, который она может формировать или требовать во время TAR-обработки. Возвращаясь к пояснению относительно управления ключами, одна из основных операций управления ключами может заключаться в том, чтобы анализировать данную TAR и формировать соответствующий список шаблонов типов ключей, которые могут детализировать тип и технические требования каждого ключа, который может требоваться в успешной обработке данной TAR. Табл. 3506 перечисляет по меньшей мере девять типов для типов ключей, заданных в SDFT.

Табл. 4300 показывает преобразование каждой операции превращения, которая может требовать ключа и соответствующего типа ключа или "keytyp", что она может требовать посредством процесса запутывания/распутывания. Шаблон ключей может иметь несколько атрибутов, ассоциированных с каждым типом ключа, таких как, но не только, длины ключей или "дл. ключа". Для краткости и упрощения иллюстрации, можно указывать симметричный ключ с длиной в 256 битов как имеющий шаблон ключей, который может представляться как "symmetric keylen=256" или "symmetric 256", но в фактической реализации может использовать любые доступные механизмы структуры данных на языке программирования, чтобы сохранять такие значения организованным способом. В Python v3.6, возможная структура для шаблона ключей может представляться посредством матрицы словарей, при этом каждая словарная статья в матрице сохраняет один шаблон ключей с каждым атрибутом, соответствующим словарному ключу, и значением атрибута, соответствующим значению, ассоциированному с этим ключом в словаре. В SDFT, все шаблоны ключей могут представлять собой временные структуры и могут подвергаться повторяющимся регенерациям через превращение с проверкой ключей, и может быть необязательным постоянно сохранять такие шаблоны ключей. Таким образом, SDFT может надлежащим образом анализировать любые ключи, вставленные в стек ключей для обработки, до позволения криптографическому превращению немедленно сбоить вследствие несовместимостей типов ключей/структур. Преобладающая тема в TOP и SDFT может представлять собой такой вид, что запутывание последовательностей манипулирования данными может не представлять собой надежный компонент в защите чувствительных рабочих данных, а вместо этого может низводиться до силы выбранного шифра и его рабочих атрибутов и/или характеристик.

Фиг. 44 показывает TAR-примеры и шаблоны ключей, сформированные из каждого. Левый столбец в табл. 4402 перечисляет TAR - пример A. Правый столбец указывает шаблоны типов ключей, сформированные для каждой команды превращения, которая может требовать криптографического ключа в качестве ввода атрибута. В этом примере, TAR A может требовать двух криптографических ключей в указываемой последовательности. Левый столбец в табл. 4404 перечисляет TAR-пример "B". Правый столбец указывает шаблоны типов ключей, сформированные для каждой команды превращения, которая требует криптографического ключа в качестве ввода. В этом примере, TAR "B" может требовать четырех криптографических ключей в указываемой последовательности. Этот процесс может быть известным как формирование шаблона ключей из TAR.

Фиг. 45 показывает TAR-примеры и шаблоны ключей, сформированные из каждого, и ожидаемый список KISS-структур, которые должны вводиться (помещаться) или формироваться (формир.). Список KISS также упоминается как стек ключей. Можно рассматривать два примера из фиг. 44 и показывать следующий этап в аспекте управления ключами вызова запутывания/распутывания. Стек ключей может ожидаться или формироваться в форме списка KISS-структур, соответствующих каждому шаблону типов ключей, как показано посредством 4510. Когда обработка TAR A достигает команды превращения "s-шифра salsa20 256", процесс может ожидать нахождение входного симметричного ключа с длиной в 256 битов в стеке ключей, как указано посредством KISS A1. Когда обработка TAR A достигает команды превращения "dign dss 1024 digestlen=512", процесс может ожидать нахождение входного 1024-битового DSA-ключа в стеке ключей, как указано посредством KISS A2. KISS-список для TAR "B" может читаться и пониматься как осуществляемый аналогичным образом. Если такой ожидаемый ключ не содержится в стеке ключей, TAR-обработка может ожидать, что сформированный ключ должен содержаться вместо этого. Это неявное формирование ключей может быть полезным для программиста, поскольку единственное требование для того, чтобы формировать любой тип приемлемого ключа для данного превращения по ключу, заключается в том, чтобы иметь возможность объявлять его в TAR. Могут не требоваться дополнительные этапы для того, чтобы формировать конкретный ключ для конкретной криптографической функции. Вызов запутывания с пустым стеком ключей может приводить к выходной NSstr-структуре, которая хранит полностью совместимый стек ключей с надлежащим образом сформирован-

ными ключами с тем, чтобы совпадать с TAR и иметь возможность складывать данные внутри. Строго рекомендуется и желательно то, что этот стек ключей, состоящий из KISS-структур, затем может сохраняться отдельно защищенным способом отдельно от сложенных данных, и/или он дополнительно может модифицироваться некоторым образом и складываться снова и защищаться с использованием TAR с криптографическим превращением, за счет этого дополнительно шифруя его. Повторяющееся шифрование и инкапсуляция стеков ключей могут быть полезными при решении проблемы, связанные со многими криптографическими ключами, чтобы управлять и защищать. TAR "B" может формировать стек ключей четырех KISS, и может быть удобным защищенно сохранять весь стек ключей в репозитории ключей; тем не менее, программист может хотеть шифровать стек ключей из четырех ключей с использованием одного ключа для удобства. Это может осуществляться посредством создания новой NSstr, вставки стека ключей из четырех ключей в поле obj данных, выбора соответствующей криптографической TAR и выполнения вызова запутывания для NSstr-структуры. Эта последовательность этапов может формировать стек ключей с одной KISS-структурой, содержащей замыкающий ключ, как сложенную NSstr-структуру, хранящую стек ключей из четырех ключей.

Фиг. 46 показывает три режима работы стека ключей в SDFT TAR-обработке: формирование (формир.), ввод (помещение) и введение (смешанный). Раздел 4600 иллюстрирует то, что может возникать, когда стек ключей 4602 является пустым в обработке TAR-примера "B". Процесс запутывания может принимать шаблон типов ключей для TAR "B" 4508 и формировать 4606 соответствующее число случайно сформированных криптографических ключей с типом и порядком, идентичными типу и порядку, содержащемуся в шаблоне типов ключей, как показано на 4604. Раздел 4610 иллюстрирует то, что может возникать, когда стек 4612 ключей вводится (помещается) или формируется (формир.) 4616 в обработку TAR-примера "B". Процесс запутывания может принимать шаблон типов ключей для TAR "B" 4508 и сверять его с предоставленным стеком 4612 ключей, чтобы проверять достоверность числа, типа и упорядочения ключей, и после этого он может обеспечивать возможность его использования во время обработки TAR "B", как показано на 4614. Раздел 4620 иллюстрирует то, что может возникать, когда стек 4622 ключей представляется в обработку TAR-примера "B" с предоставлением только одного ключа, этой KISS B3, либо также называемый частично заполненным стеком ключей, либо сценарий "отсутствующих зубов". Процесс запутывания может принимать шаблон типов ключей для TAR "B" 4508 и сверять его с предоставленным стеком 4622 ключей, чтобы проверять достоверность числа, типа и упорядочения ключей. Во время итеративной проверки достоверности каждого шаблона типов ключей по сравнению с записью стека ключей, любая пустая KISS-структура может считаться специальным типом сбоя проверки достоверности и может дополнительно истолковываться в качестве неявного превращения с формированием ключей для этого шаблона типов ключей. Процесс запутывания затем может вводить 4626 новый сформированный ключ соответствующего типа в пустую позицию стека ключей и продолжать с итерацией проверки достоверности ключей. После выполнения этого этапа, смешанный стек ключей (может упоминаться как смешение введенных и сформированных ключей, сценария отсутствующих зубов или введения ключей) может представляться и использоваться во время обработки TAR "B", как показано на 4624.

Фиг. 47 показывает иллюстрацию того, как стеки ключей могут формироваться и использоваться в жизненном цикле данных и их TAR. Использование SDFT для данных 4700 может обеспечивать возможность их итеративного превращения упорядоченным способом согласно переменному набору превращений, заданному посредством конкретной TAR 4702. TAR может быть структурирована таким образом, чтобы обеспечивать анализ типов криптографических ключей и в силу этого формировать шаблоны ключей, детализирующие число и тип ключей, необходимых посредством TAR. Шаблон ключей затем может задаваться ссылкой в композиции 4704 входного стека ключей независимо от того, могут присутствовать все, некоторые или вообще не присутствовать необходимые ключи. Когда требуемый криптографический ключ может отсутствовать, процесс композиции может формировать новый ключ для использования. TAR, данные и стек ключей затем могут пересылаться в вызов запутывания 4706, чтобы выполнять складывание структурированных данных согласно TAR. Сложенные данные затем могут сохраняться 4708 каким-либо образом. Ключи в стеке ключей могут сохраняться 4710 в отдельном защищенном местоположении. Когда к сложенные данные следует обращаться, приложение может извлекать 4712 их из их места хранения, извлекать 4714 ключи или стек ключей из его защищенной области хранения, передавать сложенные данные и стек ключей в вызов распутывания 4716 и осуществлять доступ 4702 к данным в исходной форме из структуры вывода распутывания. Это может служить признаком одного полного цикла складывания структурированных данных с превращениями. Может быть предусмотрено множество других путей для превращения и складывания любой структуры данных, но в сущности некоторая форма этого цикла может требоваться для того, чтобы завершать и полностью извлекать исходные данные в SDFT.

Хранение ключей и/или стека ключей 4710 может предусматривать складывание стека ключей с использованием криптографической TAR, чтобы защищать его с помощью меньшего количества ключей, только одного ключа и/или различных ключей. Сложенные данные стека ключей могут становиться частью другой структуры, которая может в конечном счете складываться непосредственно. Данные могут

складываться итеративно каскадным способом, чтобы компоновать внутренние структуры данных, при этом точное складывание по частям может приводить к точным шифрованиям по частям. Эта способность направлять сложные превращения криптографических данных точным, организованным и/или методическим способом может приводить к более оптимальным и/или более простым проектным решениям для защиты конфиденциальных данных с использованием более сложных схем превращения. Простота и ясность TAR-синтаксиса может приводить к лучшему пониманию операций, осуществляемых с целевыми данными посредством других.

Важное преимущество SDFT может заключаться в систематической обработке управления ключами в контексте комбинирования различных криптографических операций для данного фрагмента данных, к примеру, 4704 и 4714. Программист может быть в определенной степени освобожден от маловажных деталей формирования каждого ключа и манипулирования вручную его хранением и/или упорядочением в ходе таких процессов. В приложении криптографических функций, эти маловажные детали могут быстро нарастать, так что они становятся огромным числом небольших подробностей или атрибутов, которые приложение (в силу этого и программист) должно отслеживать, анализировать, сохранять и/или использовать. SDFT-способы могут обеспечивать возможность данному приложению отслеживать, анализировать, сохранять и/или использовать меньшее число отдельных атрибутов криптографических функций, поскольку они могут обеспечивать возможность встраивания этих атрибутов в контекст данных и/или в стек ключей, в котором они работают или который они формируют в качестве вывода, в силу чего они могут предоставлять попарное связывание сложенных данных наряду с превращениями, которые могут складывать их. Трансплантация инструкций манипулирования данными из приложения в данные может предоставлять возможность более простых приложений и/или приложений с более сложным использованием криптографических функций. SDFT может обеспечивать лучшую альтернативу для того, чтобы выражать способ структурированного криптографического программирования (SCP), как пояснено в разделе относительно NUTS.

Фиг. 48 показывает иллюстрацию операций, которые могут возникать в данных, сохраненных в NSstr-структуре. Любые данные, на которые ссылаются посредством указателя, и/или которые сохраняются в переменных 4802, могут инкапсулироваться 4812 в NSstr-структуру непосредственно, с использованием способа создания экземпляра и/или с использованием 4804 вызова метода/функции. Затем NSstr-структура 4810 может инкапсулировать TAR 4814 и его ассоциированные атрибуты 4816 при необходимости. Атрибуты могут содержать стеки ключей, дайджесты, параметры превращения и/или временные переменные. Это может предоставлять минимальный полный набор информации, необходимой для того, чтобы обрабатывать NSstr через SDFT-операцию запутывания/распутывания, чтобы выполнять TAR для данных, содержащихся внутри, с использованием атрибутов, которые могут предоставляться 4810. В TOP-диалекте, это можно называть складыванием данных. Вывод SDFT может возвращаться в качестве идентичной NSstr-структуры 4810 или NSx-структуры, такой как NSjson. Этот вывод затем может сохраняться 4820 некоторым постоянным и доступным способом, передаваться 4840 в другое вычислительное устройство с использованием способа межпроцессной связи (IPC) и/или сохраняться 4830 в другой внутренней структуре данных. Цикл может начинаться снова для сохраненных данных 4820 и 4830 в последующей точке в приложении. Для передаваемых данных 4840, цикл может инициироваться посредством приема таких пакетов 4800 данных.

Фиг. 49 показывает блок-схему последовательности операций способа SDFT-использования, чтобы итеративно складывать данные. Последовательность упрощенных схем показывает систематическое складывание данных с использованием SDFT-вызовов запутывания для N последовательных складываний данных. NSstr-структура 4900, содержащая, по меньшей мере, данные 4902 и TAR 4904, может складываться посредством вызова 4906 запутывания, чтобы формировать выходные данные 4908, которые могут модифицироваться и/или дополнительно инкапсулироваться в NSstr-структуру 4920, содержащую, по меньшей мере, данные 4922 и TAR 4924, которая может складываться посредством вызова 4926 запутывания, чтобы формировать выходные данные 4928, которые могут модифицироваться и/или дополнительно инкапсулироваться в NSstr-структуру 4940, содержащую, по меньшей мере, данные 4942 и TAR 4944, которая может складываться посредством вызова 4946 запутывания, чтобы формировать выходные данные 4948, которые могут модифицироваться и/или дополнительно инкапсулироваться, ..., этот процесс может проходиться итеративно по мере необходимости 4950. Следует отметить, что в этой сложной последовательности складываний структурированных данных, любая TAR на любом этапе может модифицироваться отдельно от кода приложения посредством простой модификации инструкций TAR, сохраненных в некотором текстовом файле или его эквиваленте.

Эквивалентное программируемое выражение без SDFT таких итеративных инкапсуляции с возможностью последовательности превращений и/или параметрических разбросов для каждого этапа может быть сравнительно длинным, подверженным ошибкам и/или трудным в обнаружении.

Фиг. 50 показывает блок-схему последовательности операций способа SDFT-использования, чтобы итеративно раскладывать данные. Последовательность упрощенных схем показывает систематическое раскладывание данных с использованием SDFT-вызовов распутывания для N последовательных раскладываний данных. Она представляет собой точную обратную последовательность последовательности

операций по фиг. 49 и в силу этого может пониматься как таковая. Как показано ранее на фиг. 39 и 40, вызов распутывания может быть идентичным вызову запутывания за исключением подготовки TAR и состояния данных, подаваемых в него. Следует отметить, что в этой сложной последовательности раскладываний структурированных данных, дополнительные обратные TAR могут не требоваться для того, чтобы достигать разворачиваний. Все необходимые TAR, необходимые для того, чтобы распутывать каждые сложенные данные, могут считаться встроенными в сложенном конструкте. Более тесный анализ NStar-структуры 3106 показывает поле expd, заданное в качестве "списка формы с расширенным набором TAR-команд". Он может представлять собой фундаментальное свойство обратимости в SDFT: вывод этапов 3912 и 4012 TAR-подготовки может формировать полный набор рабочих команд превращения, не имеющих ссылок метки и любых других внешних ссылок, и может считаться полным описанием превращений, которым могут подвергаться превращенные данные. Он может рассматриваться в качестве статического снимка экрана TAR-набора для сложенных данных, в силу этого гарантируя то, что надлежащее раскладывание может выполняться для сложенных данных независимо от изменений TAR-определений во внешнем местоположении. Это может подразумевать, что файлы TAR-определений могут расти во времени с большим числом TAR-определений, но объем хранения рабочего TAR-определения может сокращаться посредством SDFT-процесса таким образом, чтобы сохранять его обратимость независимо от изменений таких внешних файлов определений (что может не быть рекомендуемой практикой). Это проектное решение может стимулировать систематический способ разрешать временную совместимость сохраненных данных лучшим способом.

Фиг. 51 показывает иллюстрацию SDFT-API/библиотеки и различных типов файлов TAR-определений, к которым она может иметь доступ. TAR-определение может существовать во множестве форм, таких как, но не только, текстовые файлы, Nut-контейнеры, зашифрованные файлы, базы данных, серверные процессы, и/или в рабочем запоминающем устройстве. TAR может задаваться в любое время программистом в качестве индивидуально настраиваемого TAR-определения в вызове запутывания и в силу этого может представлять собой временную TAR. Для этих TAR-определений, которые могут постоянно сохраняться, схема 5100 может иллюстрировать различные их формы, но может не быть ограничена посредством показанных TAR-определений. Стандартные TAR 5102 могут представлять собой TAR-определения, которые могут предоставляться в качестве комплекта наряду с установкой SDFT-библиотеки для любого спаривания ОС/языка. Скрытые TAR 5104 могут представлять собой TAR-определения, которые могут представлять собой индивидуально настраиваемыми TAR-определения, которые могут существовать только в местоположениях с ограниченным доступам и/или быть доступными посредством разрешения с помощью выражения. Они могут представлять собой предпочтительный способ TAR-определений в закрытой сети или установке настраиваемого приложения. Использование скрытых TAR может поддерживаться скрытым даже в выводе запутывания, и расширенная форма TAR не может считаться встроенной в такие сложенные данные, а просто ссылка на нее посредством TAR-метки. Обязательство поддерживать скрытые TAR может лежать на администраторах таких групп, поскольку данные, складываемые со скрытыми TAR, могут не обязательно содержать набор превращений, требуемых для того, чтобы раскладывать их. Скрытые TAR могут казаться знакомыми в качестве эквивалентного способа запутывания последовательностей манипулирования данными внутри программы.

Локальные пользовательские TAR 5106 могут представлять собой TAR-определения, которые могут представлять собой индивидуально настраиваемые TAR-определения, к которым может осуществляться доступ только согласно привилегиям учетной записи пользователя или программиста. Они могут быть временными или развивающимися TAR, которые программист может формулировать для постоянного добавления в одну из форм хранения TAR-определений позднее. Удаленные TAR 5108 могут представлять собой TAR-определения, к которым может осуществляться доступ с/без доступа по разрешению из удаленного сервера или места хранения. Такая топология может требоваться вследствие ограниченного локального хранилища или вследствие политики централизации TAR-определений ключей в централизованно управляемую область. Это также может представлять собой способ постоянной проверки того, могут или нет стандартные TAR-определения быть последними версиями. Защищенные TAR 5110 могут представлять собой TAR-определения, которые могут быть расположены в любом соответствующем, доступном месте, но могут шифроваться только для авторизованного доступа. Отдельный процесс аутентификации и авторизации, возможно, должен обходиться успешно, чтобы получать доступ к защищенным TAR. Другая форма защищенной TAR может сохраняться в Nut-контейнере, который может требовать надлежащего ключа(ей), чтобы получать доступ в него. Встроенные сложенные TAR 5112 могут представлять собой расширенные TAR-определения, сохраняемые наряду со сложенными данными из вызова запутывания.

Фиг. 52 показывает примерный Python-сценарий, чтобы выполнять складывание данных вручную. Фиг. 53 показывает SDFT-пример TAR-определения и его использования в Python-сценарии. Фиг. 52 и 53 могут совместно показывать пример того, как SDFT отличается от более прямого программируемого подхода с использованием Python v3.6. Эти примерные Python-сценарии могут иллюстрировать существенные отличия в базовых вызывающих последовательностях для каждой задачи под рукой с использованием каждой технологии. Начнем с набора дискретизированных данных 5210. Операции, которые сле-

дует выполнять для данных, могут указываться в задачах 5220, выражаемых на простом языке, как показано в строках 02-06. Обычно они могут вводиться в саму программу в качестве строк комментариев для удобочитаемости. Секция 5250 показывает фактический Python-код для того, чтобы выполнять задачи, и секция 5260 показывает обратную обработку задач для того, чтобы восстанавливать исходные данные 5210.

При использовании SDFT, набор 5310 данных является идентичным 5210. Секция 5320 выражает задачи 5220 в качестве TAR-определения, помеченного как "test_a70". Секция 5350 запутывает данные и записывает сложенные данные в файл. Секция 5360 читает сложенные данные из файла и распутывает их.

Предусмотрено 18 строк Python-кода для фиг. 52 и только 8 строк кода на фиг. 53. Очевидно, что любые изменения типов и числа превращений данных могут затрагивать обе секции 5250 и 5260. Способ на фиг. 52 требует от программиста поддерживать несколько переменных, последовательность задач и/или надлежащий вызов каждой функции или метода. Обратный процесс на 5260 требует от программиста удостоверяться в том, что все операции вызываются в корректном обратном порядке, и параметры подаются корректно для каждого вызова функции или метода. Любые изменения задач на 5220 могут приводить к программированию изменений секций 5250 и 5260. Любые дополнительные задачи на 5220 могут приводить к дополнительным программным строкам в секциях 5250 и 5260. Дополнительные временные переменные могут создаваться и использоваться по мере необходимости для этих добавлений или изменений задач.

В SDFT-способе на фиг. 53, любые изменения задач могут непосредственно отражаться в TAR 5320. В силу этого любые дополнительные модификации превращения могут варьировать только длину этой секции. Строки 10 и 14 вызова запутывания и распутывания остаются неизменными. Процесс обращения на 5360 TAR 5320 не должен обязательно указываться за рамками исходного TAR-определения на 5320. Фактически, секции 5350 и 5360 могут оставаться невозмущенными для любого выбранного TAR-определения, за исключением строки 10, в которой метка TAR-определения указывается в вызове метода запутывания.

С точки зрения удобочитаемости и понятности выполняемых задач, читатель может предпочитать TAR 5320 по сравнению с фактическим программным кодом в секциях 5250 и 5260. Задачи, указываемые на 5220, не представляют собой код и могут обычно выражаться как комментарии в Python-коде. Все изменения программного кода в секциях 5250 и 5260 должны вручную координироваться с комментариями программистом, в противном случае может возникать путаница, если другой программист должен пытаться разобраться в коде с неточными комментариями, и наоборот. TAR 5320 может считаться самоописываемой понятным и компактным способом.

Данные, сохраненные посредством строк 15-16 в секции 5250, не имеют встроенных метаданных, описывающих то, как они могут превращаться. Технология превращений аппаратно кодируется в секциях 5250 и 5260 в качестве фактического кода. Любые такие данные, записываемые таким образом, могут полностью зависеть от существования идентичного или аналогичного кода для их надлежащего извлечения и восстановления. Эти секции кода или их эквиваленты должны поддерживаться навсегда для данных, которые они превращают с возможностью восстановления навсегда. Они могут представлять собой эквивалент скрытого TAR-способа.

Данные, сохраненные посредством строки 11 в секции 5350, могут содержать встроенное расширенное TAR-определение, которое может превращать сложенные данные. Технология превращений может спариваться со сложенными данными, за счет этого обеспечивая их транспортабельность. Восстанавливаемость сложенных данных может считаться независимой от кода, который создает их 5350 и 5360. Любой код, который может надлежащим образом обрабатывать встроенное TAR-определение в сложенных данных, может восстанавливать исходные данные. Этот тип функциональности может предоставлять возможность лучшей временной совместимости для изменения последовательностей превращений во времени, поскольку устаревшие сложенные данные могут самоописывать и в силу этого самопредписывать то, как они могут восстанавливаться.

Фиг. 54 показывает блок-схемы динамической TAR-коммутации в одном сеансе связи. В TOP-подходе, высокоуровневый протокол связи может рассматриваться в качестве пересылки превращенных данных из одного процесса вычисления в другой. Поскольку превращения могут обеспечивать множество наиболее часто используемых криптографических функций, они могут использоваться для того, чтобы создавать защищенные сообщения для IPC. Теоретически, каждое сообщение может превращаться и складываться с использованием различной TAR. Каждое различное TAR-определение может рассматриваться как собственный протокол по современным стандартам определений протокола. При использовании SDFT, TAR могут коммутироваться динамически в расчете на сложенное сообщение между двумя приложениями, как проиллюстрировано на фиг. 54. Любая смесь TAR-источников, как показано и описано на фиг. 51, может использоваться при условии, что каждое приложение может иметь доступ к этим источникам TAR-определений. Обширный набор встроенных, сложенных метаданных, таких как, но не только, KISS-структуры в качестве замочных скважин, указывающих точный идентификатор ключа для каждого ключа, необходимого во встроенном криптографическом превращении, могут обеспечивать

возможность протоколам связи на основе SDFT предлагать безопасность на более сложном и потенциально более защищенном уровне.

TOP-анализ и способы, которые могут приводить к инфраструктуре, называемой SDFT, могут обеспечивать возможность сохраненным данным содержать собственный портативный набор инструкций, который может формировать их. Эта инфраструктура может задавать складывание данных и может предоставлять технологии и/или варианты осуществления, чтобы складывать данные с использованием концептуально и логически согласованного способа обработки обратимого превращения, выразимого в качестве записи с результатами аудита превращений (TAR), которая может встраиваться в сохраненные данные организованным способом. Результирующие сложенные данные затем могут модифицироваться некоторым способом и после этого могут многократно складываться по мере необходимости, чтобы достигать требуемого результата по форме приложения или данных. За исключением описания TAR в качестве языка программирования, они представляют набор совместного манипулирования данными в точной форме, которая может обеспечивать бесконечные варьирования последовательностей превращений и/или бесконечные варьирования атрибутов превращения в данной TAR и/или атрибутах. SDFT может предоставлять возможность переменного определения объема для наборов данных, аналогично способу, которым языки программирования изолируют локальные переменные с использованием концептов и технологий определения объема. Через TOP протокольные разбросы могут рассматриваться в более высококонцептуальном конструкте, который может приводить к данным, которые могут быть само описываемыми и возможно могут быть доступными и читаемыми из широкого спектра приложений, которые могут осуществлять доступ к его технологиям через доступную SDFT-библиотеку, адаптированную для их окружения программирования. Кроме того, эти характеристики, которые могут внедряться в сложенные данные, могут предоставлять возможность динамической коммутации протоколов в одном сеансе связи или в одном объекте сохраненных данных. TOP-подход может использоваться в качестве фундаментального компоновочного блока для NUTS-экосистемы и в композиции Nut-контейнера. NUTS может полностью реализовываться независимо от SDFT, но это может быть нецелесообразным.

NUT-идентификатор

Проектное NUTS-решение может предоставлять идентифицируемость данных независимо от местоположения. Это может требовать универсально уникального идентификатора (UUID), но он может не быть достижимым гарантированным способом без некоторой формы централизации, в силу чего можно остановиться на понятии практически уникального идентификатора с достаточной длиной и энтропийными свойствами, чтобы предоставлять низкую вероятность коллизий идентификатора. Фиг. 55 показывает блок-схему последовательности операций способа для примера процесса для формирования Nut-идентификатора 5500. Здесь локальное устройство 5502 может выполнять приложение, который может активировать функцию, чтобы формировать практически уникальный идентификатор из фрагментов данных, таких как, но не только, пользовательские атрибуты 5504, атрибуты 5506 окружения и/или случайные атрибуты 5508. Пользовательские атрибуты 5504 могут включать в себя элементы данных, такие как, но не только, информация входа пользователя в учетную запись, идентификатор группы, идентификатор компании, идентификатор пользователя и/или хеш-пароля пользователя. Атрибуты 5506 окружения могут включать в себя элементы данных, такие как, но не только, MAC-адрес, IP-адрес, информация устройства, системное время, информация ОС, пути к каталогам и/или файлы каталогов, синхронизированные по атомным часам значения времени, GPS-синхронизированные значения времени, объявленные переменные окружения, идентификатор подпроцесса, среда выполнения CPU, IMEI-номер, телефонный номер, имя приложения и/или идентификатор процесса. Случайные атрибуты 5508 могут включать в себя элементы данных, такие как, но не только, счетчики сеансов, UUID, счетчики тактовых циклов, случайно сформированные числа, перемещение мыши, активность клавиатуры, состояние файловой системы, хеши частичной или полной области экрана, время непрерывной работы процесса, время непрерывной работы ОС и/или длительность сеанса. Эти фрагменты данных могут собираться и сохраняться в структуре 5510 идентификаторов, которая затем может преобразовываться в последовательную форму с использованием JSON или альтернативных технологий маршалинга. После этого результирующая двоичная строка может хешироваться 5520 с использованием алгоритма хеширования, такого как SHA512 (из SHA2-семейства хеш-алгоритмов, опубликованных в FIPS PUB 180-2 посредством NIST в 2001 году) или альтернативного способа хеширования, который может формировать практическую уникальность с предлагаемой минимальной длиной 512 битов для того, чтобы снижать вероятность коллизий идентификатора. Двоичный хеш может кодироваться в текстовой строке на основе base64 (или альтернативной схемы кодирования) для портативности и удобочитаемости 5514, что позволяет формировать текстовую строку с длиной на 86 символов больше или меньше. Схема кодирования может содержать любой способ, который может приводить к печатаемой и человекочитаемой форме, и может подтверждаться посредством множества языков программирования и программных систем в качестве текстовой строки. В зависимости от модальности, в которой может вызываться функция, результирующая кодированная хеш-строка может сверяться на предмет дублирования с любым доступным кэшем 5516 Nut-идентификаторов. Если может возникать коллизия значений идентификаторов, в таком случае процесс может повторяться с новыми случайными атрибутами 5508 до тех пор, пока не сможет формироваться

некофликтующий идентификатор; коллизии могут предполагаться как редкие ситуации. Выходная строка этой логической операции может называться Nut-идентификатором 5518.

Этот процесс может вызываться локально в выполняющейся программе или может реализовываться на серверном приложении, постоянно размещающемся локально или удаленно обслуживающем запросы клиентского приложения на предмет новых Nut-идентификаторов. Возможное преимущество реализации на основе серверной модели может заключаться в ее способности осуществлять доступ к большим кэшам существующих Nut-идентификаторов, чтобы выполнять сверку, и может формировать Nut-идентификатор с более низкой вероятностью коллизии. Сверка дублирования Nut-идентификаторов не является обязательной, поскольку длина хеша и надлежащим образом собираемые компоненты данных в структуре 5510 идентификаторов могут предоставлять достаточную энтропию. Может быть предусмотрен общий концепт разделения для некоторых или всех цифровых инфраструктур, таких как Интернет с IPv4/IPv6-адресами, доменами, иерархиями каталогов и группами управления доступом. Аналогично, Nut-идентификатор может быть практически уникальным, но он, вероятно, может использоваться в контексте отсека, конструируемого посредством внешней системы или взаимосвязи, и в силу этого вероятности коллизий могут быть на множество порядков величины меньшими, чем математические вероятности, предлагаемые посредством перестановок при данной длине битов Nut-идентификатора. В случаях, если может требоваться другая длина, это может осуществляться посредством замены SHA512-хеша на альтернативный хеш-алгоритм модульным параметризованным способом специалистами в данной области техники.

С учетом процесса, посредством которого практически уникальный идентификатор может формироваться в форме Nut-идентификатора, что может идентифицироваться посредством него? В NUTS-диалекте, это может быть известно как штамповка Nut-идентификаторов. Может быть предусмотрено по меньшей мере две структуры в NUTS, которые могут согласованно штамповаться с Nut-идентификаторами: узлы замка и Nut-контейнеры. Nut-идентификатор, назначаемый узлу замка, может называться идентификатором замка. Nut-идентификатор, назначаемый Nut-контейнеру, может называться Nut-идентификатором. Узел замка может представлять собой внутренний компоновочный блок Nut-контейнера. Узел замка может представлять собой самодостаточный автономный замыкающий механизм, который может защищать свои рабочие данные, известные как мешок. Nut-контейнер может представлять собой структуру данных, состоящую из одного или более узлов замка. Следовательно, Nut-контейнер может хранить любую посылку или посылки данных полностью либо ее часть. Nut-контейнеры могут использоваться во всем NUTS-окружении, чтобы идентифицировать практически уникальным способом часть или все ассоциированное программное обеспечение, данные и/или аппаратные средства, представленные в двоичной форме. Последствие штамповки Nut-идентификатора может заключаться в том, что каждый Nut-контейнер может уникально идентифицироваться, что подразумевает то, что каждая посылка данных, сохраненная в Nut-контейнере, может уникально идентифицироваться посредством этого Nut-идентификатора независимо от того, где может физически находиться Nut-контейнер.

Фиг. 56 показывает упрощенный схематический вид Nut-структуры данных. Эта схема может подчеркивать использование и относительные размещения идентификаторов замков и Nut-идентификаторов в Nut-структуре данных. Конкретные идентификаторы 5614-5622 замков могут назначаться в этом Nut-контейнере, и они могут быть различными значениями. Узлы 5604-5612 замка могут, соответственно, идентифицироваться посредством идентификаторов 5614-5622 замков. В типичном формировании Nut-структуры данных, как в этом примере, Nut-контейнер 5602 может представлять собой группу узлов замка, организованных в графовидную структуру данных, называемую графом замков. Конкретный Nut-контейнер 5602 может идентифицироваться посредством своего Nut-идентификатора 5634, который может сохраняться в мешке 5626 узла 5606 замка, и Nut-идентификатор может считаться рабочими данными этого узла замка, которые могут отличаться от рабочих данных Nut-контейнера, который может сохраняться в одном или более других мешков узлов замка. Каждая структура узла 5604-5612 замка может содержать область рабочих данных, называемую мешком 5624-5632. Она показывает взаимосвязь между Nut-контейнером и его Nut-идентификатором, а также то, где можно находить эти элементы, сохраненные в типичном Nut-контейнере.

Фиг. 57 показывает примеры взаимосвязей между Nut-идентификаторами, именами путей и/или рабочими данными. Может быть предусмотрено несколько узлов замка в Nut-контейнере, которые могут использоваться для того, чтобы сохранять метаданные Nut-контейнера, метаданные относительно рабочих Nut-данных и/или рабочих Nut-данных. Части метаданных могут сохраняться в мешках различных узлов замка в Nut-контейнере. 5702 показывает сценарий, в котором могут быть предусмотрены два различных вида рабочих Nut-данных D1 и D2, сохраненных в различных Nut-контейнерах, идентифицированных посредством Nut-идентификаторов A3 и B1 соответственно. Двусимвольные Nut-идентификаторы используются в качестве иллюстрации, даже если заранее может указываться то, что Nut-идентификатор может представлять собой кодирование base64512-битового хеша, который может формировать текстовую строку с длиной до 86 символов. Nut-контейнеры A3 и B1 также имеют различные имена путей в файловой NTFS-системе. 5704 показывает два различных Nut-контейнера, имеющие

идентичное имя файла, но различные имена путей. 5706 показывает две копии идентичного Nut-контейнера с идентичными именами файлов в различных каталогах. 5708 показывает две копии Nut-контейнера с различными именами файлов, сидящими в идентичном каталоге. Это может не представлять собой полный перечень некоторых или всех перестановок этих атрибутов, но он может демонстрировать гибкость наличия элемента метаданных, постоянно ассоциированного с каждыми рабочими данными, такого как Nut-идентификатор.

Данные, встраиваемые внутри Nut-файла, который может идентифицироваться посредством ассоциированного Nut-идентификатора, могут обуславливать новый признак этой технологии: способность автоматически создавать динамические имена файлов на основе параметризованных правил в метаданных. Имя файла может представлять нормальную идентифицирующую строку для файла, а также сформулированную сводку его других атрибутов, таких как, но не только, дата и время модификации и/или число записей в течение дня. Это может обеспечивать более точный и удобный способ идентификации файла и его состояния во времени без необходимости тщательно исследовать нормально скрытые атрибуты, а также необходимости смотреть на свойства файла в приложении для просмотра каталогов. Это также может обеспечивать возможность встраивания атрибутов файлов и данных в контейнер, хранящий файл, а не основываться на возможностях захвата атрибутов файловой системы, которые могут варьироваться в зависимости от файловой системы. Пример: пользователь может создавать Nut-контейнер с Nut-идентификатором #234, который может сохранять текстовый документ, текстовый документ может всегда идентифицироваться посредством Nut-идентификатора #234, но пользователь может устанавливать динамическое имя файла, содержащее "базовое имя + дата последней модификации + количество записей в течение дня", к примеру, "diary 20151115 1.txt". В этот же день, когда он сохраняет на диск после модификации свой бит, имя файла может демонстрировать "diary 20151115 2.txt", и старое имя файла более не может существовать в каталоге. Эта технология может автоматически создавать новое имя файла, которое может указывать некоторую информацию состояния сохраненных данных. Свойства Nut-идентификатора, который может быть практически уникальным и может быть отдельным от обозначений на основе имени пути + имени файла, могут обеспечивать возможность реализации такого признака без внешних ссылок. Одно из преимуществ такого признака может заключаться в часто используемом способе копирования и архивации предыдущих состояний рабочего документа с отметкой даты. Автор может находить каталог, заполненный с файлом для каждого дня, с которым может работать в своем документе. При использовании динамического способа на основе имен файлов, он может иметь только один Nut-файл в каталоге с отметкой даты прошлого времени, когда он записывал в него. Аспект сохранения предыстории (состояний) ручного способа может сохраняться в самом Nut-контейнере с использованием функции Nut-предыстории, представленной в последующем разделе. Этот концепт Nut-идентификатора, представляющего собой основные идентификационные данные контента, может использоваться в дальнейшем посредством NUTserver для того, чтобы выполнять операции репликации и синхронизации для рассеянных Nut-контейнеров.

Графы замков и узлы замка NUTS-технология может быть нацелена на хранение, защиту и управление доступом данных в многоуровневом, интегрированном, модульном и/или итерационном подходе, который может задаваться как структурированное криптографическое программирование (SCP). Общее проектное решение внутренних элементов Nut-контейнера может описываться и задаваться, и затем заданная структура может далее описываться подробно. Некоторые признаки могут описываться многоуровневым способом, и после этого описание интеграции может предоставляться, чтобы показывать то, как отдельные признаки могут работать совместно. SDFT может использоваться во всем проектном NUTS-решении для того, чтобы улучшать организацию сложных криптографических структур и систематическое встраивание атрибутов, ассоциированных с каждой сложенной структурой данных. В различных вариантах осуществления можно показывать то, как SDFT обеспечивает возможность реализации проектных SCP-решений с относительной простотой по сравнению с эквивалентными ручными способами.

Может быть предусмотрено четыре различных технологии, которые могут управлять доступом Nut-контейнера: замочная скважина, изменяемый замок, управление доступом к слою (SAC) и/или управление Nut-доступом (NAC). Некоторые или все эти технологии частично или полностью могут разделяться на уровни и/или интегрироваться между собой новыми способами в Nut-контейнере, который может предоставлять полную функциональность опорной системы мониторинга интернализированным и/или независимым способом. Эти четыре уровня могут быть осуществлены в сложной структуре данных, называемой узлом замка, который может проектироваться с возможностью быть модульным, замкнутым и/или связываемым.

Замочная скважина может представлять собой структуру данных, которая может подтверждать любое число ключей шифрования, каждый из которых может иметь ассоциированную зашифрованную карту ключей. Вариант осуществления не ограничен типами ключей шифрования, которые он может в данный момент распознавать и подтверждать: фразовый пароль, симметричный ключ и пара асимметричных ключей. Любой простой или сложный способ или любой процесс, который может указывать последовательность битов в качестве секретного ключа, могут интегрироваться в замочную скважину. Зашифро-

ванная карта ключей может содержать несколько наборов ключей, один набор для каждого уровня управления доступом в Nut-контейнере: изменяемый замок, SAC и/или NAC.

Изменяемый замок может предоставлять различные типы замыкающих механизмов в нормализованной структуре, которая может защищать данные в узле замка. Эти изменяемые замки могут содержать OR-замок, MAT-замок, SS-замок, XOR-замок и хеш-замок. Это раскрытие сущности не ограничено этими предварительно заданными типами замка, но может расширяться или сужаться, чтобы приспосабливать любую соответствующую схему замыкания, которая может нормализоваться в означенную структуру.

Управление доступом к слою может упорядочивать доступ проникновения в отдельные узлы замка в графе замков. Этот признак может обуславливать свойство в Nut-контейнерах, называемое непрозрачностью градиента, которое может представлять собой способность для Nut-контейнера, чтобы обеспечивать возможность просмотра различных уровней метаданных с учетом соответствующих атрибутов доступа.

Управление Nut-доступом или NAC может использовать технологии криптографического управления доступом на основе ролей (RBCAC) для того, чтобы точно управлять модификациями и аутентификациями внутренних элементов Nut-контейнера.

Структурированное криптографическое программирование может представлять собой проектирование структур данных, которые могут обеспечивать возможность легко достижимых и гибких взаимодействий между различными технологиями, чтобы выражать множество моделей доступа. Механизмы обеспечения безопасности могут быть полностью осуществлены в зашифрованных данных и их ассоциированных шифрах, в силу чего могут отсутствовать внешние зависимости приложений от управления доступом Nut-контейнера, такого как опорный монитор. В некоторых вариантах осуществления, узел замка может использоваться отдельно, чтобы защищать данные уровня поля в любой части рабочих данных. Внутренние элементы Nut-контейнера могут потенциально использовать множество ключей шифрования для того, чтобы осуществлять конкретную модель обеспечения безопасности.

Nut-контейнер может представлять собой структуру данных в форме направленного графа, называемую графом замков, состоящим из узлов, называемых узлами замка. Каждый узел замка может идентифицироваться посредством идентификатора замка, который может создаваться посредством идентичной функции для формирования Nut-идентификатора, в силу чего они могут иметь идентичные характеристики. Узлы замка могут сохраняться в хешированной матрице, к которой можно обращаться посредством их идентификаторов замков. Каждый узел замка может иметь указатели, связующие с другими идентификаторами замков, или нулевой указатель. При использовании общепринятых технологий программируемого извлечения и обхода графа, граф замков может извлекаться из хешированной матрицы узлов замка. Узел замка, который не имеет других узлов замка, указывающих на него, может представлять собой узел замка с замочной скважиной (запись или внешний узел замка). Узел замка, который может иметь нулевой указатель, может представлять собой предельный узел замка графа замков и может сохранять рабочие данные Nut-контейнера или ссылку на рабочие данные. Узел замка может иметь несколько узлы замка, связующие с ним. В большинстве обстоятельств, узел замка не связывается обратно с более ранним узлом замка в графе замков или с собой. Круговая связующая ссылка может быть необычной, но может приспосабливаться через индивидуально настраиваемое программирование для настраиваемых Nut-контейнеров, если гарантирована такая структура.

Часть, если не вся структура данных, описанная в данном документе, которая поддерживает функциональности Nut-контейнера, может реализовываться с использованием сложных структур данных в рамках выбранного языка программирования. Если функциональная библиотека SDFT доступна для выбранного языка программирования, она может легко применяться, чтобы складывать и инкапсулировать все без исключения применимые сложные структуры данных либо их подчасти для того, чтобы минимизировать код для манипулирования данными, прояснять способы манипулирования данными, уменьшать вероятность ошибок кодирования и использовать преимущество подразумеваемых SDFT-признаков, встроенных в каждую сложенную структуру данных.

Следует отметить, что вследствие датацентрического характера этого раскрытия сущности, большинство схем типа блок-схемы последовательности операций способа могут представлять собой смесь традиционных элементов блок-схемы последовательности операций способа, смешиваемых с компонентами данных, которые могут упоминаться как диаграммы потоков данных или блок-схемы обработки данных. Кроме того, переплетающийся характер уровней проектирования на основе узлов замка может затруднять раскрытие логических операций их компонентов полностью линейным способом без осуществления прямых ссылочных утверждений, в силу чего некоторое повторное чтение может требоваться со стороны читателя.

Фиг. 58 является вариантом осуществления графа 5800 Nut-контейнеров или замков, содержащего две логических секции с использованием многоцелевых аспектов модульных узлов замка: Nut-замок 5802 и Nut-части 5804. Секция Nut-замка 5802 графа замков может предоставлять возможность конструирования сложных криптографически связанных замков для данного Nut-контейнера с использованием одного или более узлов замка. В данный момент предусмотрено пять типов узлов замка, заданных в

этом раскрытии сущности, соответствующих упомянутым пяти типам изменяемых замков: OR-замок, MAT-замок, SS-замок, XOR-замок и хеш-замок. Каждый тип узла замка может означать тип внутреннего замыкающего механизма изменяемых замков, который может использоваться в самом центре конкретного узла замка, чтобы защищать ключи шифрования в области хранения и другие метаданные и параметры узлов замка. Превращения замков, раскрытые на фиг. 30, могут представлять собой вариант осуществления изменяемых замков и могут использоваться в компоновке узла замка. Успешное отмыкание и обход части Nut-замка 5802 графа замков может приводить к секции Nut-частей 5804 графа 5800 замков. Может быть предусмотрено несколько узлов замка, которые содержат Nut-части 5804: шерсть 5820, верхний кожный покров 5822, нижний кожный покров 5824, вита 5826, бейл 5828 и/или хвост 5830. Nut-части 5804 могут содержать рабочие Nut-данные 5830 и/или метаданные 5820-5828. Число и тип Nut-частей для графа замков могут варьироваться в зависимости от типа данных, которые Nut-контейнер может сохранять, и/или проектирования Nut-контейнеров для некоторых требуемых поведений и характеристик. В этом примере, отмыкание узла 5806 (5816) замка с замочной скважиной может приводить к надлежащим ключам шифрования, которые могут вставляться в замочную скважину под первичный ключ связанного 5818 узла 5820 замка. Отмыкание узла 5820 замка может приводить к надлежащим ключам шифрования, которые могут вставляться в замочную скважину под первичный ключ связанного узла 5822 замка. Отмыкание узла 5822 замка может приводить к надлежащим ключам шифрования, которые могут вставляться в замочную скважину под первичный ключ связанного узла 5824 замка. Отмыкание узла 5824 замка может приводить к надлежащим ключам шифрования, которые могут вставляться в замочную скважину под первичный ключ связанного узла 5826 замка. Отмыкание узла 5826 замка может приводить к надлежащим ключам шифрования, которые могут вставляться в замочную скважину под первичный ключ связанного узла 5828 замка. Отмыкание узла 5828 замка может приводить к надлежащим ключам шифрования, которые могут вставляться в замочную скважину под первичный ключ связанного узла 5830 замка. Узел 5830 замка может связываться с нулевым указателем, в силу чего он может представлять собой предельный узел замка или крайний внутренний уровень этого графа замков или Nut-контейнера. Отмыкание узла замка может содержать распутывание сложенной структуры данных, представляющей узел замка, с использованием SDFT-способов (раскладывания). Каждый узел замка может содержать множество сложенных структур данных, при этом действие отмыкания узла замка может быть эквивалентным раскладыванию применимых структур данных.

Фиг. 59 показывает упрощенную Nut-семантику 5900 варианта осуществления графов замков, содержащего логические секции Nut-замка 5902 и Nut-частей 5904. Этот пример исследует Nut-замок 5902, содержащий четыре узла 5908, 5910, 5912 и 5916 замка. Узлы 5908-5912 замка могут представлять собой узлы 5906 замка с замочной скважиной этого Nut-контейнера, поскольку некоторые или все из них могут представлять собой обращенные наружу внешние узлы и могут подтверждать внешние ключи шифрования, называемые первичными ключами. Пользователь может иметь первичные ключи, ассоциированные с одним или более этих узлов замка с замочной скважиной. Nut-идентификатор Nut-контейнера, сохраняющего первичный ключ в качестве своих рабочих данных, может выступать в качестве идентификатора ключа, который может автоматически согласовываться с идентификатором, помечающим замочную скважину, которой он принадлежит, из числа узлов 5906 замка с замочной скважиной. Ключи-фразовые пароли могут идентифицироваться посредством идентификаторов ключей или текстовой строки, которая может хранить или не может хранить вопрос в качестве его идентификатора. Сложные многоуровневые фразовые пароли могут конструироваться с использованием надлежащих идентификаторов замочных скважин и частей Nut-метаданных в форме открытого текста с соответствующими списками вопросов. Связывания между узлами замка, такими как 5914 и 5918, могут открываться аналогичным образом, когда успешно отомкнутый узел замка может формировать выведенный ключ(и) с идентификатором. В этом конкретном примере, отмыкание любого из узлов замка с замочной скважиной может раскрывать надлежащие ключи шифрования, которые могут вставляться в замочную скважину связанного 5914 узла 5916 замка. Далее с этого момента, отмыкание узлов, содержащих Nut-части 5904, может продолжаться аналогично процессу, описанному для Nut-частей 5804. Это конструирование Nut-замка 5902 может передавать характер компоновочных блоков узлов замка и гибкость его комбинаций посредством демонстрации того, что могут существовать три различных пути для того, чтобы отмыкать рабочие данные 5900 Nut-контейнера, при этом каждый путь требует удовлетворения различных условий, чтобы продолжать процесс отмыкания.

На фиг. 60 узел 6000 замка может представлять собой структуру данных, содержащую следующие секции: параметры 6002, ввод 6006, карты 6008 ключей, изменяемый замок 6012, извлеченный ключ 6016, набор 6020 ключей, мешок 6024 и/или вывод 6026. Секция 6002 параметров может хранить метаданные узла замка, идентификатор 6030 замка и зашифрованные строки карт 6010 ключей, извлеченный ключ 6014, набор 6018 ключей, мешок 6022 и ц-подписи упомянутых зашифрованных строк, созданных посредством соответствующих ролевых ключей доступа (прямая ссылка может описываться в пояснении для элемента 8334 фиг. 83) для узла замка. Принципы проектирования могут быть аналогичными потоку в графе замков с отмыканием каждой секции, что может приводить к ключам, которые могут помогать открывать следующую секцию, но в таком случае каждый компонент в узле замка может предоставлять

конкретную функцию. Ц-подписи для зашифрованных строк могут использоваться читателями (ролью для доступа) для того, чтобы аутентифицировать конкретную секцию до попытки дешифрования. Ц-подписи могут создаваться писателями (ролью для доступа) конкретной секции с использованием зашифрованной строки секции, когда могут возникать некоторые модификации, которые следует сохранять, или указывать то, что надлежащий держатель ключей доступа писателя формирует ц-подпись. Кроме того, каждая из вышеуказанных зашифрованных строк может быть осуществлена посредством использования SDFT-способов, чтобы складывать структуры данных с использованием TAR, содержащих криптографические превращения. С учетом числа и разнообразия зашифрованных строк, описанных в этом разделе, SDFT-способы могут радикально уменьшать нагрузку по управлению криптографически связанными атрибутами программистом при кодировании.

Замочные скважины

На фиг. 61 секция 6006 ввода узла замка может предоставлять две различных замочных скважины: замочную скважину 6102 под первичный ключ и замочную скважину 6104 под ключ доступа. Структурно, замочная скважина 6102 под первичный ключ может подтверждать любое число криптографических ключей, содержащих четыре различных типа ключей: симметричный, асимметричный открытый, асимметричный закрытый и фразовый пароль. Замочная скважина 6104 под ключ доступа может подтверждать типы симметричных ключей и/или ключей-фразовых паролей. Замочная скважина под первичный ключ и замочная скважина под ключ доступа могут внутренне использовать одну или более KISS-структур данных, как показано на фиг. 34, каждая из которых работает в режиме замочных скважин (ima="keyhole"), чтобы представлять замочную скважину для каждого уникального ключа, который это может подтверждать.

Фиг. 62 показывает один криптографический ключ 6202, который может иметь ассоциированный идентификатор ключа, тип ключа и атрибуты 6206 ключей и также может обозначаться в качестве первичного ключа. Идентификатор ключа может представлять собой любую идентифицирующую строку. Первичный ключ и любые другие ключи, упомянутые в этом раскрытии сущности, могут внутренне представляться посредством KISS-структуры данных, как показано на фиг. 34, работающей в режиме ключей (ima="ключ") с полем key->value, заполненным ключом, и другими совпадающими полями атрибутов, заполненными требуемым образом. Замочная скважина 6204 под первичный ключ может подтверждать первичный ключ 6202, который может дешифровать зашифрованную карту 6208 ключей. Дешифрованная карта 6240 ключей может представлять собой структуру, которая может содержать три секции: основной ключ 6210, ключи 6212 слоя и набор 6214 ключей доступа (AKS). Структура 6210 основного ключа может содержать симметричный ключ или зубец, который может называться основным ключом, дату истечения срока действия/время для первичного ключа 6202, таймер обратного отсчета для первичного ключа и/или инструкции с действиями после истечения срока первичного ключа. Симметричный ключ или зубец может использоваться посредством изменяемого замка узла замка. Для узла замка с замочной скважиной, структура карт ключей дополнительно может хранить ключи 6212 слоя и/или AKS 6214. Ключи 6212 слоя могут хранить набор ключей, которые должны вставляться в узлы замка слоев графа замков, в те узлы замка, которые могут идентифицироваться посредством своего обозначения слоя. AKS 6214 может хранить набор ключей, которые должны вставляться в собственную замочную скважину 6104 под ключ доступа для узла замка с замочной скважиной. Зашифрованная карта 6208 ключей может представлять собой сложенную SDFT-структуру данных, которая может содержать структуры основного ключа 6210, ключей 6212 слоя и набора 6214 ключей доступа (AKS).

Фиг. 63. показывает блок-схему последовательности операций способа для процесса вставки ключей для любого узла замка и для любого ключа шифрования. Этап 6304 может представлять собой поиск по некоторым или всем перечисленным узлам замка в Nut-контейнере на предмет данного ключа шифрования и его ассоциированного идентификатора ключа. После того, как ключ шифрования вставляется в надлежащую замочную скважину 6304, этап 6306 может пытаться дешифровать и развертывать зашифрованную карту ключей для этого ключа. Дешифрование и разворачивание зашифрованной карты ключей могут быть эквивалентными распутыванию сложенной зашифрованной SDFT-карты ключей для таких вариантов осуществления.

После успешного отмыкания и разворачивания зашифрованной карты 6208 ключей для узла замка с замочной скважиной, 1) ключи слоя могут вставляться в замочную скважину под первичный ключ каждого узла замка, совпадающую с обозначением слоя, содержащимся в каждой секции параметров узла замка, 2) ключи отмыкания наборов ключей атрибутов доступа (AAKSUK) из набора ключей доступа (AKS) могут вставляться в замочную скважину под ключ доступа узла замка. Это отмыкание (или распутывание) первичных ключей может возникать для стольких первичных ключей, сколько может вставляться в узел замка, после чего можно иметь набор дешифрованных (или разложенных) карт ключей, совместно составляющих набор основных ключей для возможного применения посредством изменяемого замка узла замка.

Фиг. 64 показывает пример, в котором три первичных ключа 6402-6406 могут вставляться в замочную скважину 6400 под первичный ключ. Каждый ключ (→) может согласовываться с его идентифицирующим идентификатором ключа и может вставляться в слот в хешированной матрице или структуре

замочных KISS-скважин. Тип ключа может указывать тип ключа, такой как, но не только, симметричный, асимметричный открытый, асимметричный закрытый и фразовый пароль. В некоторых вариантах осуществления Nut-контейнера, пользователь может указывать любой тип ключа, который может иметь соответствующую технологию шифрования, надлежащим образом модуляризированную для NUTS-интеграции. Эти технологии шифрования ключей могут включать в себя сканирование отпечатков пальцев, сканирование радужной оболочки глаз, отпечатки ладоней, отпечатки голоса, шаблоны почерка, распознавание лиц, DNA-характеристики, физические устройства-ключи, аппаратно-защищенные ключи, программные/аппаратные ключи по протоколу с "нулевым знанием" и/или NFC-ключи. Если вставляется асимметричный закрытый ключ, к примеру, который может использоваться в RSA-2048, он может представлять как открытую, так и закрытую части, открытая часть может извлекаться из закрытой части и может использоваться для того, чтобы шифровать зашифрованную карту ключей первичного ключа, в силу чего операция дешифрования может требовать представления закрытого асимметричного ключа. Как явно показано для одного ключа (→), вставленного в одну замочную скважину 6402, его зашифрованная карта 6412 ключей может дешифроваться с использованием технологии шифрования по типам ключей, чтобы раскрывать структуру 6430 карт ключей, которая может содержать три отдельных набора ключей 6432, 6434 и 6436. Этот этап дешифрования может осуществляться для каждого ключа 6404 и 6406, чтобы формировать соответствующие надлежащие наборы 6440 и 6450 карт ключей. Каждый этап дешифрования также может быть эквивалентным распутыванию сложенной SDFT-структуры для таких вариантов осуществления. Для типа ключа-фразового пароля, ключ может представлять собой фразовый пароль, и атрибуты ключей могут указывать функцию извлечения фразового пароля, которую следует использовать, и соответствующие параметры для функции, содержащей число итераций, которые выполнять для того, чтобы формировать симметричный ключ, который может дешифровать зашифрованную карту ключей. Для вариантов осуществления с использованием SDFT, такие атрибуты фразового пароля также могут согласовываться с соответствующей TAR, чтобы осуществлять доступ к соответствующим превращениям с извлечением с совпадающими атрибутами. Если представлять пример в перспективе со схемой 6000 узлов замка, секция 6006 ввода может содержать замочную скважину 6400 под первичный ключ, зашифрованные карты 6010 ключей могут представляться посредством 6410, и секция карт 6008 ключей может представляться посредством 6420.

Изменяемые замки

Следующая часть узла замка может представлять собой изменяемый замок, как показано в элементе 6012 по фиг. 60. Изменяемый замок может представлять собой замыкающий механизм, который может помогать защищать контент узла замка, сохраненного в мешке 6024. Изменяемый замок может обеспечивать возможность узлу замка использовать любые из нескольких различных типов криптографических технологий замыкания, знакомых специалистам в данной области техники. Например, эти различные типы замка могут содержать OR-замок, MAT-замок, XOR-замок, хеш-замок и/или SS-замок. Это может осуществляться посредством нормализации вводов и/или выводов каждого способа замыкания, так что они вписываются в общую модель потоков данных, в силу чего все способы замыкания могут прозрачно меняться между собой. Аналогично, замочная скважина под первичный ключ и структуры карт ключей могут выступать в качестве нормализаторов данных для числа ключей и типов ключей, протекающих в изменяемый замок. Узел замка может быть отпечатан с набором параметров 6002, указывающих то, какой изменяемый замок он может реализовывать 6030. После того, как это значение задается, узел замка может редко изменять эти настройки, хотя может быть возможным повторно вводить ключи и/или сбрасывать узлы замка посредством RAT (владельца Nut-контейнера). SDFT-библиотека описывает вариант осуществления изменяемых замков, перечисленный на фиг. 30, и его прилагаемую спецификацию, которая может использоваться в этом разделе для удобства, но использование превращения замков не представляет собой обязательное требование для того, чтобы удовлетворять этой функциональности узла замка.

При продолжении обхода узла замка на фиг. 64, на котором имеется три основных ключа 6432, 6442 и 6452, можно исследовать то, как его изменяемый замок может работать на фиг. 65. Изменяемый замок 6502 может защищать извлеченный ключ 6506 (DK) посредством его шифрования в качестве зашифрованного извлеченного ключа 6504 (eDK). Некоторые или все основные ключи 6432, 6442 и 6452 могут представлять собой типы симметричных ключей или типы ключей-зубцов и могут подаваться в изменяемый замок 6502. В зависимости от типа изменяемых замков, который может указываться в секции 6002 и 6030 параметров узла замка, соответствующая функция изменяемых замков может вызываться для того, чтобы выполнять операцию шифрования/дешифрования для DK или eDK. Фиг. 65 показывает операцию дешифрования eDK 6504 в DK 6506 посредством изменяемого замка 6502, которая может использовать основные ключи 6432, 6442 и 6452. Фиг. 66 показывает операцию шифрования DK 6506 в eDK 6504 посредством изменяемого замка 6502 с использованием основных ключей 6432, 6442 и 6452. В варианте осуществления с использованием SDFT, DK может представлять собой данные, которые защищаются посредством TAR с использованием превращения замков посредством складывания данных; в силу этого раскладывание такой структуры раскрывает извлеченный ключ, содержащийся внутри.

Таблица на фиг. 67 обобщает некоторые характеристики ключей упомянутых изменяемых замков.

Как может подразумевать термин "изменяемый замок", любая технология замыкания, которая может нормализоваться в эту модель, может добавляться в качестве дополнительного типа изменяемых замков. Альтернативно, любая технология замыкания может исключаться. Таблица на фиг. 30 может соответствовать таблице на фиг. 67 и показывает то, как SDFT может осуществлять проектирования на основе изменяемых замков в своих превращениях замков.

Секция 6030 метаданных узла замка может представлять собой общий компонент, который может быть предусмотрен в некоторых или во всех изменяемых замках. Могут быть предусмотрены различные ц-подписи (цифровые подписи) секций узла замка, которые могут создаваться посредством соответствующего ролевого ключа доступа (ARK), такого как 6040-6048 (прямая ссылка). Некоторые из всех этих ц-подписей могут создаваться Nut-владельцем, который может быть любым, хранящим ролевой ключ доступа на уровне корневого доступа (RAT), в частности, закрытый RAT-ключ, через свой AKS. Все с допустимым первичным ключом могут иметь открытый RAT-ключ, который может обеспечивать возможность им аутентифицировать различные RAT-ц-подписи для всего узла замка, чтобы удостоверяться в том, что Nut-компоненты не могут быть скомпрометированы. На схемах, иногда открытый RAT-ключ может упоминаться как ключ RAT-читателя, и закрытый ключ может упоминаться как ключ RAT-писателя. Ниже в этом документе, дополнительные пояснения относительно уровня управления Nut-доступом могут исследовать, указывать и/или прояснять эти признаки с большей глубиной. Как упомянуто выше в разделе относительно SDFT и TAR, ц-подписи зашифрованных данных могут составлять часть TAR-спецификации сложенной структуры данных, которая может встраивать защищенные данные, их ц-подпись и TAR, которая создает их. Это явно подразумевает, что систематическое использование SDFT в узле замка может быть преимущественным в отношении рабочей нагрузки программистов.

OR-замок на фиг. 68, также известный как OR-замок, представляет собой изменяемый замок, который может подтверждать любое число симметричных ключей шифрования, называемых основными ключами 6808, и может систематически пытаться дешифровать 6814 eDK 6806 с использованием симметричного криптографического шифра, такого как AES-256 или альтернативный шифр. Секция 6002 параметров может указывать способ шифрования, который следует использовать для этой логической операции, или предпочтительную TAR при использовании SDFT-способов. Первое успешное дешифрование eDK может формировать извлеченный ключ 6816 (DK) и может приводить к успешному отмыканию OR-замка. До попытки дешифрования в любом изменяемом замке, ц-подпись eDK 6804 может аутентифицироваться с использованием eDK 6806 и ключа 6802 RAT-читателя. Если аутентификация завершается удачно 6810, то процесс дешифрования может продолжаться, иначе может вызываться ошибка 6830, и попытка может прекращаться. Основные ключи 6808 могут представлять собой ключи, идентичные таким ключам, как, но не только, симметричные 256-битовые ключи. В этой компоновке, сущность OR-замка может изолироваться и нормализоваться в структуры замочных скважин и изменяемых замков, чтобы обеспечивать их модульность. В сложенной структуре, этап аутентификации может составлять часть TAR и может неявно осуществляться посредством действия распутывания.

Фиг. 69 иллюстрирует операцию шифрования OR-замка с точки зрения RAT-писателя или Nut-владельца. Она может принимать 6902 любой основной ключ и может выполнять 6920 операцию шифрования для DK 6906 с использованием соответствующего шифра, чтобы формировать eDK 6908. Затем с использованием своего ключа 6904 RAT-писателя, eDK 6908 и соответствующего алгоритма 6922 снабжения ц-подписью, она может создавать 6910 ц-подпись eDK, которая может сохраняться в секции 6044 параметров узла замка. SDFT-способы могут складывать множество этих атрибутов компактным способом наряду с eDK в один объект данных, который должен сохраняться в секции параметров. Процесс шифрования для Не-RAT-членов узла замка может быть простым; либо они могут стирать контент запоминающего устройства для хранения приложений узла замка, поскольку они не могут вообще создавать подлинную ц-подпись, что подразумевает то, что они не могут успешно изменять свой контент и снабжать его ц-подписью, либо они могут использовать уже дешифрованный DK 6908, и они могут шифровать релевантный контент узла замка, но могут оставлять eDK 6910 нетронутым, поскольку не может изменяться ничего, что может быть релевантным для eDK-ц-подписи. Это может показывать то, что только RAT-писатели могут иметь возможность заменять значение DK 6906 или повторно вводить его ключи. При использовании SDFT-способов, Не-RAT-члены узла замка могут решать оставлять исходные сложенные данные, содержащие eDK, в секции параметров, и стирать разложенную структуру, хранящую DK.

MAT-замок на фиг. 70, также известный как замок-матрешка, каскадный замок или AND-замок, представляет собой изменяемый замок, который может подтверждать фиксированное число симметричных ключей шифрования, называемых основными ключами 7006, и может последовательно дешифровать eDK 7022 с использованием каждого основного ключа 7008 в порядке возрастания с использованием соответствующего криптографического шифра 7014, такого как AES-256 или альтернативный шифр. Секция параметров может указывать точный шифр, который следует использовать для этой логической операции, и число основных ключей, которые могут требоваться, или предпочтительную TAR при использовании SDFT-способов. Успешные упорядоченные итеративные дешифрования eDK 7022 могут формировать DK 7024 и могут приводить к успешному отмыканию MAT-замка. До попытки дешифрова-

ния в любом изменяемом замке, ц-подпись eDK 7004 может аутентифицироваться с использованием eDK 7022 и ключа 7002 RAT-читателя. Если аутентификация завершается удачно 7010, то процесс дешифрования может продолжаться, в противном случае может вызываться ошибка 7030, и попытка может прекращаться. В этой компоновке, сущность замка-матрешки может изолироваться и нормализоваться в структуры замочных скважин и изменяемых замков, чтобы обеспечивать их модульность. В сложенной структуре, этап аутентификации может составлять часть TAR и может неявно осуществляться посредством действия распутывания.

Фиг. 71 иллюстрирует операцию шифрования MAT-замка с точки зрения RAT-писателя или Nut-владельца. Она может принимать некоторые или все представленные основные ключи 7102 и может сортировать 7110 их в порядке убывания. Далее она может итеративно выполнять операции 7112 шифрования для DK 7120 с использованием соответствующего шифра, чтобы формировать eDK 7122. После этого, с использованием своего ключа 7124 RAT-писателя, eDK 7122 и соответствующего алгоритма 7118 снабжения ц-подписью, она может создавать ц-подпись eDK 7126, которая может сохраняться в секции 6044 параметров узла замка. SDFT-способы могут складывать множество этих атрибутов компактным способом наряду с eDK в один объект данных, который должен сохраняться в секции параметров. Процесс шифрования для не-RAT-членов узла замка может быть простым; либо они могут стирать контент запоминающего устройства для хранения приложений узла замка, поскольку они не могут вообще создавать подлинную ц-подпись, что подразумевает то, что они не могут успешно изменять свой контент и снабжать его ц-подписью, либо они могут использовать уже дешифрованный DK 7120, и они могут шифровать релевантный контент узла замка, но могут оставлять eDK 7126 нетронутым, поскольку не может изменяться ничего, что может быть релевантным для eDK-ц-подписи. Это может показывать то, что только RAT-писатели могут иметь возможность заменять значение DK 7120 или повторно вводить его ключи. При использовании SDFT-способов, Не-RAT-члены узла замка могут решать оставлять исходные сложенные данные, содержащие eDK, в секции параметров, и стирать разложенную структуру, хранящую DK.

XOR-замок на фиг. 72, также известный как XOR-замок, представляет собой изменяемый замок, который может подтверждать фиксированное число (>1) симметричных ключей шифрования, называемых основными ключами 7206, и может формировать вычисленный ключ посредством последовательного применения XOR-операций 7224 к каждому основному ключу 7208 в порядке 7222 возрастания. После этого он может пытаться дешифровать 7228 eDK 7210 с использованием вычисленного ключа из 7224 с помощью соответствующего шифра, такого как AES-256 или альтернативный шифр. Секция 6030 параметров может указывать точный шифр, который следует использовать для этой логической операции, и число основных ключей, которые могут требоваться, которое может составлять не меньше двух ключей, или предпочтительную TAR при использовании SDFT-способов. Успешное дешифрование eDK 7210 может формировать DK 7212 и может приводить к успешному отмыканию XOR-замка. До попытки дешифрования в любом изменяемом замке, ц-подпись eDK 7204 может аутентифицироваться с использованием eDK 7210 и ключа 7202 RAT-читателя. Если аутентификация завершается удачно 7220, то процесс дешифрования может продолжаться, в противном случае может вызываться ошибка 7230, и попытка может прекращаться. В этой компоновке, сущность XOR-замка может изолироваться и нормализоваться в структуры замочных скважин и изменяемых замков, чтобы обеспечивать их модульность. В сложенной структуре, этап аутентификации может составлять часть TAR и может неявно осуществляться посредством действия распутывания.

Фиг. 73 иллюстрирует операцию шифрования XOR-замка с точки зрения RAT-писателя или Nut-владельца. Она может принимать некоторые или все представленные основные ключи 7302 и может сортировать их в порядке 7320 возрастания. Далее она может итеративно выполнять XOR-операции 7322 для основных ключей 7304, чтобы формировать вычисленный ключ, который может использоваться для того, чтобы шифровать 7326 DK 7306, чтобы формировать eDK 7308. Ключ 7310 RAT-писателя, eDK 7308 и соответствующий алгоритм 7328 снабжения ц-подписью могут использоваться для того, чтобы создавать ц-подпись eDK 7312, которая может сохраняться в секции 6044 параметров узла замка. SDFT-способы могут складывать множество этих атрибутов компактным способом наряду с eDK в один объект данных, который должен сохраняться в секции параметров. Процесс шифрования для не-RAT-членов узла замка может быть простым; либо они могут стирать контент запоминающего устройства для хранения приложений узла замка, поскольку они не могут вообще создавать подлинную ц-подпись, что подразумевает то, что они не могут успешно изменять свой контент, либо они могут использовать уже дешифрованный DK 7306, и они могут шифровать релевантный контент узла замка, но оставлять eDK 7312 нетронутым, поскольку не может изменяться ничего, что может быть релевантным для eDK-ц-подписи. Это может показывать то, что только RAT-писатели могут иметь возможность повторно вводить ключи DK 7306. При использовании SDFT-способов, Не-RAT-члены узла замка могут решать оставлять исходные сложенные данные, содержащие eDK, в секции параметров, и стирать разложенную структуру, хранящую DK.

Хеш-замок на фиг. 74, также известный как хеш-замок, представляет собой изменяемый замок, который может подтверждать фиксированное число симметричных ключей шифрования, называемых ос-

новными ключами 7406, и может создавать вычисленный ключ посредством конкатенации 7424 некоторых или всех основных ключей, представленных в конкретном порядке 7422, и затем он может применять алгоритм 7426 хеширования к строке. После этого он может пытаться дешифровать 7428 eDK 7410 с использованием вычисленного ключа с помощью соответствующего криптографического шифра, такого как AES-256 или альтернативный шифр. Секция 6030 параметров может указывать точный шифр и хеш, которые следует использовать для этих логических операций, число необходимых основных ключей и/или порядок сортировки основных ключей или предпочтительную TAR при использовании SDFT-способов. Успешное дешифрование eDK 7410 может формировать DK 7412 и может приводить к успешному отмыканию хеш-замка. До попытки дешифрования в любом изменяемом замке, ц-подпись eDK 7404 может аутентифицироваться с использованием eDK 7410 и ключа 7402 RAT-читателя. Если аутентификация завершается удачно 7420, то процесс дешифрования может продолжаться, в противном случае может вызываться ошибка 7430, и попытка может прекращаться. В этой компоновке, сущность замка с хешированием может изолироваться и нормализоваться в структуры замочных скважин и изменяемых замков, чтобы обеспечивать их модульность. В сложенной структуре, этап аутентификации может составлять часть TAR и может неявно осуществляться посредством действия распутывания.

Фиг. 75 иллюстрирует операцию шифрования хеш-замка с точки зрения RAT-писателя или Nut-владельца. Она может принимать представленные основные ключи 7502 и может сортировать их в порядке 7520 возрастания, затем может конкатенировать 7522 их и после этого может формировать вычисленный ключ посредством выполнения операции хеширования 7524 дли них. Этот вычисленный ключ может использоваться для того, чтобы шифровать 7526 DK 7506, и может формировать eDK 7510. Ключ 7508 RAT-писателя, eDK 7510 и соответствующий алгоритм 7528 снабжения ц-подписью могут использоваться для того, чтобы создавать ц-подпись eDK 7512, которая может сохраняться в секции 6044 параметров узла замка. SDFT-способы могут складывать множество этих атрибутов компактным способом наряду с eDK в один объект данных, который должен сохраняться в секции параметров. Процесс шифрования для не-RAT-членов узла замка может быть простым; либо они могут стирать контент запоминающего устройства для хранения приложений узла замка, поскольку они не могут вообще создавать подлинную ц-подпись, что подразумевает то, что они не могут успешно изменять свой контент, либо они могут использовать уже дешифрованный DK 7506, и они могут шифровать релевантный контент узла замка, но оставлять eDK 7512 нетронутым, поскольку не может изменяться ничего, что может быть релевантным для eDK-ц-подписи. Это может показывать то, что только RAT-писатели могут иметь возможность повторно вводить ключи DK 7506. При использовании SDFT-способов, Не-RAT-члены узла замка могут решать оставлять исходные сложенные данные, содержащие eDK, в секции параметров, и стирать разложенную структуру, хранящую DK.

SS-замок на фиг. 76, также известный как замок по принципу разделения секретов или схема разделения секретов по принципу Шамира, представляет собой изменяемый замок, который может подтверждать k из n основных ключей 7606, каждый из которых может представлять собой различный зубец или квоту разделения секретов, и где $1>p+1\leq k\leq n$, и p+1 может представлять собой минимальное число требуемых ключей, называемое пороговым значением. Чтобы восстанавливать секретный ключ, некоторые или все зубцы из дешифрованных карт 7606 ключей могут предоставляться в соответствующий шифр 7622 по принципу разделения секретов, такой как схема разделения секретов по принципу Шамира или альтернативный шифр. Восстановление может завершаться удачно, если некоторые или все зубцы могут быть допустимыми, и может быть предусмотрено достаточное их число. После этого он может пытаться дешифровать 7624 eDK 7608 с использованием восстановленного секретного ключа с помощью соответствующего криптографического шифра, такого как AES-256 или альтернативный шифр. Секция 6030 параметров может указывать точные шифры, которые следует использовать для операций разделения секретов и шифрования, а также число квот (n) и пороговое количество (p+1) для шифра по принципу разделения секретов и/или предпочтительную TAR при использовании SDFT-способов. Успешное дешифрование eDK 7608 может формировать DK 7610 и может приводить к успешному отмыканию SS-замка. До попытки дешифрования в любом изменяемом замке, ц-подпись eDK 7604 может аутентифицироваться с использованием eDK 7608 и ключа 7602 RAT-читателя. Если аутентификация завершается удачно 7620, то процесс дешифрования может продолжаться, в противном случае может вызываться ошибка 7630, и попытка может прекращаться. В этой компоновке, сущность замка по принципу разделения секретов может изолироваться и нормализоваться в структуры замочных скважин и изменяемых замков, чтобы обеспечивать их модульность.

В сложенной структуре, этап аутентификации может составлять часть TAR и может неявно осуществляться посредством действия распутывания.

Фиг. 77 иллюстрирует операцию шифрования SS-замка с точки зрения RAT-писателя или Nut-владельца, который может шифровать узел замка в первый раз, либо который может выполнять процесс повторного ввода ключей изменяемого замка. Новый секретный ключ шифрования, K, может формироваться 7720, и затем требуемое число квот (зубцов) может создаваться из K с использованием соответствующей технологии разделения секретов, которая может указываться в параметрах 6030. Эти зубцы затем могут сохраняться в качестве основных ключей 7702. На этапе 7724, ключ K может шифровать eDK

7706 формирования DK 7704. Ключ 7708 RAT-писателя, eDK 7706 и соответствующий алгоритм 7726 снабжения ц-подписью могут использоваться для того, чтобы создавать ц-подпись eDK 7710, которая может сохраняться в секции 6044 параметров узла замка. SDFT-способы могут складывать множество этих атрибутов компактным способом наряду с eDK в один объект данных, который должен сохраняться в секции параметров. Процесс шифрования для не-RAT-членов узла замка может быть простым; либо они могут стирать контент запоминающего устройства для хранения приложений узла замка, поскольку они не могут вообще создавать подлинную ц-подпись, что подразумевает то, что они не могут успешно изменять свой контент, либо они могут использовать уже дешифрованный DK 7704, и они могут шифровать релевантный контент узла замка, но могут оставлять eDK 7706 нетронутым, поскольку не может изменяться ничего, что может быть релевантным для eDK-ц-подписи. Это может показывать то, что только RAT-писатели могут иметь возможность повторно вводить ключи DK 7704. При использовании SDFT-способов, Не-RAT-члены узла замка могут решать оставлять исходные сложенные данные, содержащие eDK, в секции параметров, и стирать разложенную структуру, хранящую DK.

Описания изменяемых замков и иллюстрации их различных логических операций могут показывать то, как узел замка может использовать замочные скважины 6102 под первичный ключ в секции 6006 ввода, зашифрованные карты 6010 ключей, карты 6008 ключей, изменяемые замки 6012, зашифрованные извлеченные ключи 6014 и/или извлеченные ключи 6016 для того, чтобы создавать надежную структуру данных, которая может предоставлять возможность нормализации и модуляризации различных технологий замыкания таким образом, что замена одной на другую может требовать некоторых изменений параметра 6030 и/или повторного ввода ключей. Нормализация различных способов замыкания может гарантировать то, что первичные ключи пользователя для Nut-контейнера могут быть нетронутыми, и то, что однопользовательский первичный ключ может использоваться во множестве различных технологий замыкания в различных Nut-контейнерах, без ведома пользователя, и какие технологии замыкания могут считаться подходящими для защиты конкретных рабочих Nut-данных. Подчеркиваются разделы, в которых SDFT-способы могут оказываться преимущественными в варианте осуществления некоторых из этих сложных структур данных. Вот некоторые примеры. OR-замок может обеспечивать возможность нескольким пользователям получать доступ к мешку узла замка: она может представлять собой форму группового доступа, или один из ключей может представлять главный ключ. MAT-замок, XOR-замок или хеш-замок может гарантировать то, что определенное число ключей может присутствовать для того, чтобы распутывать его мешок: чувствительный корпоративный секрет может требовать от двух конкретных высших руководителей, предоставления своих соответствующих секретных ключей, чтобы просматривать этот контент. SS-замок может требовать присутствия минимального числа секретных ключей для того, чтобы получать доступ к его мешку: к корпоративной платежной системе может осуществляться доступ посредством минимального числа авторизованных сотрудников, но она не может управляться одним человеком.

Посредством разделения каждой замочной скважины под первичный ключ с ее соответствующей картой ключей, карта ключей может содержать атрибуты для первичного ключа, такие как, но не только, дата/время истечения срока действия, таймер обратного отсчета и/или действие по истечению срока действия. Если какой-либо из атрибутов истечения срока действия отключается, то соответствующее действие по истечению срока действия может задаваться с возможностью выполняться по истечению срока действия первичного ключа. Например, типичное действие по истечению срока действия может заключаться в том, чтобы удалять карту ключей для первичного ключа. Удаление карты ключей не может создавать помехи другим зарегистрированным первичным ключам узла замка с замочной скважиной вследствие своего разделенного проектного решения. Повторная вставка истекшего первичного ключа более не может распознаваться в качестве допустимого ключа, поскольку может не быть совпадающей карты ключей для него. Конечно, такие удаления первичного ключа должны осуществляться тщательно в отношении типа используемого изменяемого замка: удаления могут быть приемлемыми для OR-замков и некоторых SS-замков, но они могут быть контрпродуктивными для MAT-замков, XOR-замков и хеш-замков, поскольку они могут создавать ситуацию принудительного запирания для этого узла замка.

Взаимодействие сложных структур данных, которые могут использовать множество криптографических технологий в целях защиты своего контента множеством способов и уровней, может вызывать значительные проблемы в сведениях по реализации вследствие необычно большого числа переменных атрибутов, требуемых и/или сформированных в расчете на криптографическую операцию. Именно при таких обстоятельствах полезность и элегантность SDFT сияет и позволяет обеспечивать удобные способы организации и структуры для того, чтобы помогать в преодолении таких сложностей реализации. Например, одно аутентифицированное шифрование данных может требовать сохранения в каком-либо месте следующих атрибутов: тип ключа, длина ключа, тип шифрования, режим шифрования, вектор инициализации, идентификатор ключа, дополнение, тип дополнения, длина дополнения, длина блока, цифровая подпись или MAC-строка (дайджест) с ключом, совпадающий идентификатор ключа для дайджеста, длина дайджеста, длина ключа дайджеста, способ создания дайджестов. Если умножить это на каждую операцию шифрования, описанную в спецификации узла замка, представленной выше (узел замка имеет еще несколько компонентов, которые поясняются в последующих разделах), и может быть преду-

смотрено огромное количество атрибутов для отслеживания. Во многих случаях, прикладные программисты и проектировщики могут иметь сведения по таким сложным положениям и трудностям и могут решать упрощать процесс кодирования посредством выбора небольшого количества способов шифрования и ассоциированных значений атрибутов и их использования в ходе реализации глобальным способом. Такие упрощения могут приводить к нежелательным последствиям, таким как, но не только, меньшая безопасность, меньшая гибкость, меньшее число признаков, большее число несовместимостей и машинный код, который может быть сложнее поддерживать или модифицировать.

Слой

Фиг. 78 показывает блок-схему Nut-контейнера 7800 (графа замков), подчеркивающую использование ключей слоя. Каждому узлу замка в секции 7804 Nut-частей может назначаться идентификатор слоя. Узлы 7820-7824 замка представляют собой идентификатор A слоя, узел 7826 замка представляет собой идентификатор B слоя, узел 7828 замка представляет собой идентификатор C слоя, и узел 7830 замка представляет собой идентификатор D слоя. Обозначение слоев может быть произвольным, но может придерживаться шаблона группировки различных Nut-частей посредством чувствительности к конфиденциальности: чем глубже слой, тем более чувствительными могут быть данные, содержащиеся в узле замка. Посредством точного использования средств управления доступом к слою (SAC), можно реализовывать непрозрачность градиента Nut-контейнера. В качестве иллюстрации, идентификаторы слоев, проиллюстрированные на фиг. 78, представляют собой просто буквы, но на практике может быть предусмотрен любой набор идентифицируемых строк, такой как, но не только, Nut-идентификаторы (аналогично практически уникальному идентификатору из фиг. 55).

Любые узлы замка, содержащие Nut-замок 7802, могут назначаться слой. Когда узел замка с замочной скважиной Nut-контейнера 7806 надлежащим образом отмыкается или распутывается, это может раскрывать карту 7840 ключей, которая может содержать до трех наборов 7842 ключей (аналогично фиг. 62). Эта секция может концентрироваться на ключах 7850 (6212) слоя и на том, как они могут функционировать в графе замков. В этом примере, можно находить четыре ключа 7852, 7854, 7856, 7858 слоя, которые могут соответствовать слоям "A, B, C, D" соответственно. Каждый ключ слоя может сохраняться в секции ключей 7850 слоя с ассоциированным идентификатором слоя. Можно обратиться к блок-схеме последовательности операций способа, представленной на фиг. 79, которая показывает то, как может использоваться каждый ключ слоя. После того, как некоторые или все ключи слоя могут вставляться в замочные скважины под первичный ключ их совпадающих узлов замка, процесс может быть закончен, и можно ожидать продолжения обхода графа замков за пределы секции 7802 Nut-замка.

Ключи слоя могут работать в сочетании с изменяемым замком MAT-замка, как показано в некоторых или всех узлах замка в секции 7804 Nut-частей. При использовании SDFT-способов, MAT-замок может указываться посредством "превращения" MAT-замка замка в предпочтительной TAR предусмотренной секции. Каждый ключ слоя может представлять собой обязательный ключ в MAT-замке для рассматриваемого узла замка (* на фиг. 79). Если выведенный связующий ключ или ключ слоя узла замка может отсутствовать, то конкретный узел замка не может отмыкаться согласно определению MAT-замка. Следовательно, некоторые или все более глубокие слои, за исключением этого уровня, также не могут открываться. Посредством управления тем, какие ключи слоя могут сохраняться в карте 7840 ключей для первичного ключа, Nut-владелец может явно точно управлять тем, как далеко кто-то может проникать через граф замков 7860. Уровень управления доступом к слою может работать независимо от уровня управления Nut-доступом, и он может работать в сочетании со способом на основе изменяемых замков.

Способы, посредством которых могут работать SAC и замочные скважины, могут подразумевать, что если несколько клавиш могут представляться в узел замка с замочной скважиной, к примеру, 7806, могут быть предусмотрено несколько раскрываемых карт 7840 ключей и возможно несколько наборов 7850 ключей слоя, которые могут вставляться в различные узлы замка. Ключи слоя с одним идентификатором слоя могут представлять собой идентичные ключи, так что вставка идентичного ключа в узел замка, который может использовать MAT-замок, может приводить к вставке одного ключа согласно этому идентификатору, по существу идентичный ключ может несколько раз перезаписываться в замочной скважине. Это может представлять собой свойство аддитивных атрибутов доступа ключей слоя.

Ключи слоя и управление Nut-доступом (пояснены в следующем разделе) могут демонстрировать свойство или характеристику аддитивных атрибутов доступа. Вставка первичных ключей отличающихся уровней доступа в замочную скважину под первичный ключ графа замков может приводить к уровню доступа к графу замков, который может представлять комбинацию или объединение уровней доступа ко всем допустимым вставленным первичным ключам. Одно мощное использование этого свойства может заключаться в распределении ключей для данного графа замков сегментированным способом, когда комбинация первичных ключей может требоваться для того, чтобы получать очень конкретный уровень доступа в граф замков. Это может контрастировать с режимом работы, в котором первичный ключ может представлять полное изображение данного доступа для этого держателя ключей.

Управление Nut-доступом

Управление Nut-доступом или NAC представляет собой способ управления доступом с использованием структур криптографических данных, которые могут работать независимо от изменяемых замков и управления доступом к слою. NAC может использовать комбинацию управления доступом на основе ролей (RBAC) и криптографического управления доступом (CAC), которую можно называть криптографическим управлением доступом на основе ролей (RBCAC) или разрешениям на основе ключей (KBP). Наборы ключей NAC-атрибутов могут быть локализованы во внутренние элементы одного узла замка; тем не менее, могут быть предусмотрены механизмы в узле замка для того, чтобы распространять NAC-атрибуты вдоль остальной части графа замков, что может обеспечивать для держателя ключей согласованный уровень достижимости для всех ассоциированных узлов замка. Эти NAC-атрибуты могут содержаться в отомкнутом или распутанном узле замка с замочной скважиной для первичного ключа, который может вставляться из внешнего источника. Аналогично ключам слоя, ключи NAC могут демонстрировать свойство аддитивных атрибутов доступа.

KBP может развертываться с использованием известных свойств Криптографии с открытым ключом, к примеру, создающей цифровые подписи (ц-подпись) и аутентифицирующей их асимметрично на строке данных с использованием таких алгоритмов, как RSASSA-PSS (вероятностная RSA-схема подписи с приложением на основе вероятностной схемы подписи, первоначально изобретенной Белларе и Рогавеем) или альтернативный алгоритм. Базовая предпосылка KBP может представлять собой предпосылку с учетом пары закрытого/открытого ключа, держатель с закрытым ключом (писатель) может создавать цифровую подпись (ц-подпись) в посылке данных с использованием закрытого ключа писателя, и затем держатель открытого ключа (читатель) может использовать открытый ключ писателя, которым обладает читатель, чтобы аутентифицировать то, что ц-подпись создана посредством писателя в посылке данных. Если аутентификация сбоит, то что-то может быть скомпрометировано, к примеру, открытый ключ, посылка данных или ц-подписи либо часть или все из означенного. Писатель может отвечать за создание обновленной ц-подписи для целевой посылки данных при каждой ее модификации, и читатель может отвечать за аутентификацию ц-подписи и целевой посылки данных до "чтения" или дешифрования посылки данных. Этот процесс может гарантировать для читателя в достаточной степени то, что он может читать то, что может создаваться или модифицироваться посредством кого-либо, кто может иметь закрытый ключ-дубликат (писателя). В криптографическом управлении доступом на основе ролей (RBCAC), может быть предусмотрена пара асимметричных ключей для заданной роли для доступа, и "писатель" роли может получать закрытую часть ключа, а "читатель" роли может получать соответствующую открытую часть ключа. Посредством разделения набора данных посредством функции и снабжения ц-подписью каждого функционального набора данных с использованием различных пар ключей, роли для доступа могут точно задаваться и могут назначаться различным держателям ключей посредством распределения соответствующих частей ключей. NUTS RBCAC может предоставлять возможность связывания одного или более симметричных ключей с парой асимметричных ключей заданной роли, чтобы предоставлять дополнительный уровень управления целевым набором данных. Держатели связанного симметричного ключа могут дешифровать и читать целевой набор данных для этой роли. Этот связанный симметричный ключ может шифровать целевой набор данных поверх шифрования посредством симметричного ключа, раскрытого посредством отмыкания изменяемого замка, и последующих ключей в eKS. Альтернативно, существование связанного симметричного ключа может переопределять использование раскрытого ключа шифрования из eKS и может представлять собой единственный ключ для того, чтобы симметрично шифровать целевой набор данных. Эта альтернатива может быть предпочтительной для больших целевых наборов данных, поскольку она не должна шифроваться несколько раз. Связанный симметричный ключ может использоваться для того, чтобы управлять доступом на чтение к целевому набору данных.

Использование SDFT в варианте осуществления NAC может значительно упрощать кодирование в огромной степени. Шифрование и ц-подписи могут встраиваться в логически связанные TAR, подходящие для функций, которые должны выполняться, и процесс распутывания SDFT может автоматизировать большую часть подробной обработки таких операций. Любые локализованные атрибуты, ассоциированные с TAR, могут складываться вместе с целевыми данными или дополнительно складываться с другой TAR, чтобы упрощать ее защиту и хранение.

Таблица на фиг. 80 показывает пример того, как разрешения на основе ключей могут работать с тремя заданными ролями, читателями, писателями и верификаторами, и пятью ролевыми игроками: A, B, V, X и y. Все ролевые игроки, владеющие связанным симметричным ключом S, могут иметь способность шифровать или дешифровать данные с использованием симметричного ключа S. Класс писателей (COW), X и Y, может иметь способность создавать ц-подпись для зашифрованных данных с использованием асимметричного закрытого ключа R. При использовании асимметричного открытого ключа U, класс читателей (COR), A и B, может иметь способность верифицировать то, что соответствующая цифровая подпись создана посредством кого-либо из класса писателей для зашифрованных данных, и он может иметь способность дешифровать данные с использованием симметричного ключа S. Следовательно, способность создавать допустимую ц-подпись может подразумевать, что имеется способность моди-

фицировать данные, и все другие читатели могут аутентифицировать то, что ц-подпись может создаваться посредством допустимого писателя. Число заданных ролей зависит от степени детализации управления доступом, требуемой владельцем, но некоторые или все заданные роли могут использовать технологию, как описано для фиг. 80. Ролевой игрок, который обладает только асимметричным открытым ключом U, может быть известным как верификатор; верификатор может иметь возможность обходить весь Nut-контейнер, но может не иметь возможность дешифровать целевые данные, соответствующие классу ролей. Например, COR-верификатор может аутентифицировать только то, что рабочие данные Nut-контейнера могут надлежащим образом модифицироваться посредством надлежащего ролевого COW-игрока посредством использования открытого COW-ключа для ц-подписи, но она не может дешифровать рабочие данные, поскольку на не имеет копии ключа S дешифрования.

NAC может точно затрагивать и управлять просматриваемыми и модифицированными аспектами контента, за счет чего аспектами узла замка, за счет чего аспектами Nut-контейнера. Таблица, показанная на фиг. 81, перечисляет некоторые части Nut-контейнера, но может содержать большее или меньшее число частей в зависимости от необходимости: шерсть, верхний кожный покров, нижний кожный покров, вита, морда, хвост и/или бейл. Могут быть предусмотрено несколько прямых ссылок в таблице на Nut-журналы регистрации и Nut-предысторию, которые могут подробно поясняться ниже в документе. Каждая строка может представлять узел замка, и данные, задающие Nut-часть, могут храниться в мешке этого узла замка. Столбец, называемый "непрозрачностью мешка", может показывать режим шифрования мешка узла замка, который может управляться посредством метаданных узла замка. Мешок может шифроваться или нет (прозрачный) на основе настроек метаданных, которые могут упоминаться как непрозрачность мешка. Если некоторые или все Nut-части в таблице на фиг. 81 существуют в данном Nut-контейнере, то каждая Nut-часть, которая может представляться посредством узла замка, может связываться в последовательности с начала вниз с использованием указателей связывания узлов замка и связующих ключей. Обход ниже столбца этой таблицы относительно непрозрачности мешка каждой Nut-части может упоминаться как непрозрачность градиента Nut-контейнера. Держатели надлежащего внешнего первичного ключа могут получать доступ к Nut-контейнеру посредством конечного отмыкания изменяемого замка узла замка. В зависимости от SAC-настроек первичного ключа, держатель ключа быть ограничен тем, как далеко он может осуществлять обход по Nut-контейнеру. NAC может затрагивать то, каким первичным ключам могут предоставляться способность читать, модифицировать и/или аутентифицировать каждую Nut-часть, посредством тщательного размещения связанных симметричных ключей шифрования, точного использования пар асимметричных ключей и использования способов на основе цифровой подписи.

Фиг. 82 показывает таблицу, перечисляющую роли доступа на базе разрешений на основе ключей, которые могут задаваться и быть доступными для типичного Nut-контейнера. Роли для доступа не могут быть ограничены посредством этого списка, поскольку может быть большее или меньшее число ролей для доступа, заданных в зависимости от потребностей Nut-владельца. Таблица перечисляет четыре секции Nut-контейнера, которые могут идентифицироваться, но не только: бейл, вита, хвост и "Все". Секция "Все" может означать что-либо, не охватываемое явно другой ролью для доступа. Это может влечь за собой некоторые или все внутренние операции узла замка, такие как, но не только, ц-подписи для карт ключей, eDK и/или зашифрованные мешки, не указываемые посредством пары ключей, представляющей отдельную роль для доступа. Для этого пояснения, пара ключей может содержать пару асимметричных ключей и связанный симметричный ключ. Существование связанного симметричного ключа может зависеть от существования роли верификатора класса доступа. Держатель закрытого ключа "Все" может называться RAT (уровнем корневого доступа) или владельцем Nut-контейнера. Каждый первичный ключ может иметь карту ключей, которая может содержать копию открытого ключа RAT-читателя в аутентификационных целях. Бейл может храниться в мешке узла замка, хранящего рабочие данные Nut-контейнера, такого как документ. Эта пара ключей может конкретно называться классом писателей (COW) и классом читателей (COR) вследствие ее частого использования. Эта пара ключей может управлять тем, какой первичный ключ может иметь способность модифицировать рабочие данные Nut-контейнера. Аналогично, Nut-журнал регистрации может храниться в мешке части виты Nut-контейнера и может управляться посредством пары ключей регистратора/читателя журнала регистрации. Nut-предыстория может храниться в мешке части хвоста Nut-контейнера и может управляться посредством пары ключей ведущего предысторию/читателя предыстории. Роль верификатора для каждого класса ролей для доступа может иметь доступ по меньшей мере к одному открытому ключу, ассоциированному с этой ролью, чтобы аутентифицировать ц-подпись, ассоциированную с ним. Роль верификатора может подразумевать, что может быть предусмотрен связанный симметричный ключ, ассоциированный с классом ролей для доступа, к которому он не имеет доступа. Процессу обслуживания может предоставляться доступ к комбинации заданных ролей верификатора в Nut-контейнере, чтобы проверять достоверность, согласованность и/или подлинность Nut-части, но он не может читать контент защищенных данных. Спаривания ключей не ограничены этими наборами и могут расширяться или сжиматься на основе требований. Любые зашифрованные и/или незашифрованные строки узла замка могут иметь ц-подпись, созданную для них, посредством собственной конкретной пары ключей, и все узлы замка в графе замков

могут использовать этот уровень специфичности, который может приводить к экстремальному уровню степени детализации управления доступом; тем не менее, такой экстремальный уровень степени детализации управления доступом может подавлять эффективность управления доступом для таких Nut-контейнерах.

Секция параметров узла замка может указывать алгоритм цифровой подписи, который следует применять, и длину асимметричного ключа (устанавливается по умолчанию равной минимум 2048 битам для RSA-2048). Альтернативно, SDFT-использование может предоставлять возможность конкретной TAR представлять такие предпочтения, и TAR-метка может сохраняться в секции параметров вместо этого. Зашифрованный мешок узла замка, который может хранить рабочие данные Nut-контейнера, может иметь цифровую подпись не посредством RAT-писателя с использованием ключа RAT-писателя, а вместо этого посредством держателя ключей, имеющего COW-доступ, который может включать в себя RAT-писатель. Держателям первичных ключей могут предоставляться доступ к ключу RAT-читателя через их набор ключей доступа в их карте ключей узла замка с замочной скважиной и соответствующий ключ распространения атрибутов доступа (AAPK); этот ключ RAT-читателя может обеспечивать возможность любому законному держателю первичных ключей аутентифицировать любую ц-подпись в узле замка, которая может находиться в провинции RAT-центра (примерно проиллюстрированному посредством держателя первичных ключей, который может доступ к ключу RAT-писателя). Любой сбой при аутентификации RAT-ц-подписи может подразумевать то, что соответствующая строка или сложенные данные могут быть скомпрометированы, или ключ RAT-читателя может быть недопустимым, или первичный ключ может быть более не действительным, либо некоторые или все упомянутые причины. Приложение может показывать это предупреждение и может не переходить далее за его рамки, поскольку целостность Nut-контейнера может быть скомпрометирована, и маловероятно то, что дополнительные попытки дешифрования могут успешно выполняться, либо они могут приводить к показу скомпрометированных данных.

Фиг. 83 показывает то, как Nut-контейнер достигает своего начального набора NAC-ключей доступа. Начиная с узла 8300 замка с замочной скважиной, первичный ключ 8304 может вставляться в замочную скважину 8306 под первичный ключ и может дешифровать или раскладывать зашифрованную карту ключей, которая может раскрывать структуру 8310 карт ключей, может быть предусмотрен набор 8312 ключей доступа (AKS), который может содержать набор ключей, состоящих из ключей 8314 отмыкания наборов ключей атрибутов доступа (AAKSUK), которые могут быть симметричными. Каждый отдельный симметричный AAKSUK-ключ может соответствовать роли для доступа, как показано в таблице на фиг. 82. Каждый AAKSUK в AKS затем может вставляться в замочную скважину 8320 под ключ доступа в идентичной секции 8302 ввода идентичного узла замка 8300 относительно начальной замочной скважины 8306 под первичный ключ; в силу этого карта 8310 ключей может хранить набор ключей в AKS 8312, который может подаваться в собственную замочную скважину 8320 под ключ доступа. Это может представлять собой специальное свойство узлов замка с замочной скважиной (обращенных наружу узлов замка) и может не быть применимым к внутренним узлам замка в большинстве случаев. В замочной скважине 8320 под ключ доступа, каждый надлежащим образом вставленный AAKSUK 8314 может дешифровать или раскладываться, чтобы раскрывать соответствующий набор 8330 ключей атрибутов доступа (AAKS), содержащий описание 8332 роли для доступа, ролевой ключ 8334 доступа (ARK) и/или ключ 8336 распространения атрибутов доступа (AAPK). ARK 8334 может указывать часть пары ключей, которая соответствует предоставляемой роли: открытая (читатель) или закрытая (писатель). AAPK 8336 может представлять собой симметричный ключ, который может выступать в качестве AAKSUK, в замочную скважину под ключ доступа следующего связанного узла замка. Набор AAKSUK может составлять набор AAKS, которые могут задавать NAC-атрибуты доступа первичного ключа и в конечном счете его доступ в узле замка. На этой схеме, AAKS 8330 может указывать атрибуты доступа Nut-владельца, поскольку он содержит закрытый RAT-ключ и COW-ключ. Аддитивное свойство атрибута AAKSUK (и в силу этого аддитивное свойство атрибута NAC) может быть проиллюстрировано на этой схеме; может быть предусмотрен AKS 8312 для каждого первичного ключа 8304, который может вставляться в замочную скважину 8306 под первичный ключ, так что каждая вставка AAKSUK 8314 в замочную скважину 8320 под ключ доступа может быть аддитивной. Идентичные AAKSUK могут просто перезаписывать существующий AAKSUK посредством замочной скважины под ключ доступа, что может приводить к объединению уникального AAKS, когда некоторые или все представленные первичные ключи могут обрабатываться. Это может приводить к кумулятивному эффекту атрибутов доступа, когда первичные ключи отличающихся атрибутов доступа могут вставляться одновременно.

Фиг. 84 иллюстрирует то, как AAPK могут использоваться для того, чтобы распространять NAC-атрибуты по всей остальной части узлов замка в графе замков. Узел 8400 замка с замочной скважиной может надлежащим образом отмыкаться, и часть или весь AKS 8402 может вставляться в замочную скважину 8404 под ключ доступа, что может приводить к AAKS 8420. Ключи 8424 распространения атрибутов доступа (AAPK) затем могут вставляться в замочную скважину 8432 под ключ доступа следующего связанного узла замка. Следует отметить, что это может быть аналогичным способу, которым может заполняться замочная скважина под ключ доступа узла замка с замочной скважиной, но ключи исхо-

дят из связанного узла замка, а не из AKS, который может содержаться или не может содержаться в собственной замочной скважине под первичный ключ. Замочная скважина под первичный ключ внутреннего узла 8430 замка (не показана) может иметь пустой AKS в своей карте ключей за исключением RAT-ключей уровня доступа. За счет придерживания этой технологии распространения, атрибуты доступа первичного ключа могут присутствовать в каждом открытом узле замка в графе замков. Узел замка может изолировать и локализовать некоторые или все или свои механизмы внутреннего контроля, такие как наличие различных наборов AAKS, формируемого для собственного использования в узле замка, даже если роль для доступа может быть идентичной, к примеру, COW. Даже симметричные AAKSUK- и AAPK-ключи могут отличаться при условии, что они могут преобразовываться надлежащим образом. Это может представлять собой предпосылку четко определенных Nut-контейнеров, чтобы назначать RAT с полным набором AAKS для всего узла замка и для его распространения надлежащим образом по всему графу замков. Для ссылки, может быть предусмотрен полный набор AAPK и ARK, который может шифроваться посредством открытого RAT-ключа и может сохраняться в секции параметров узла замка, так что только RAT может раскрывать его, когда ему может требоваться повторно вводить ключи для узла замка.

Фиг. 85 иллюстрирует распространение атрибутов доступа с использованием AAPK из внешнего узла 8500 замка во внутренний узел 8530 замка. Схема показывает то, из какого места могут исходить различные ключи, которые должны подаваться в замочную скважину 8550 под первичный ключ и замочные скважины 8560 под ключ доступа связанного узла. Секция 8510 вывода может раскрывать связующий симметричный ключ 8512 для замочной скважины 8550 под первичный ключ связанного узла 8530 замка. AAPK 8522 может вставляться 8524 в замочную скважину 8560 под ключ доступа связанного узла 8530 замка.

Фиг. 86 показывает блок-схему последовательности операций способа для вставки ключей в замочную скважину под ключ доступа, которая может быть подробно раскрыта с использованием примеров в предыдущих разделах.

Фиг. 87 показывает таблицу разрешений на основе ключей для альтернативного варианта осуществления. Эта таблица может разворачивать таблицу, представленную на фиг. 80, посредством дополнительного задания пары (U_w, R_w) асимметричных ключей записи и симметричного ключа S_n шифрования данных для каждого экземпляра. Три ключа из фиг. 80 могут альтернативно представляться в качестве пары (U_D, R_D) асимметричных ключей ц-подписи и симметричного ключа S_0 шифрования данных по умолчанию. Дополнительные ключи могут обеспечивать возможность ARK задавать роль для доступа "только для записи", которая может записывать в мешок узла замка, но не может читать другие части мешка. Роль "только для записи" может иметь доступ к ключам R_D, U_D, U_w и S_n. Когда роль "только для записи" хочет сохранять сообщение T_n в мешок узла замка, она может создавать одноэкземплярный симметричный ключ S_n шифрования, чтобы шифровать T_n, формирующий зашифрованное сообщение E_n. Затем один экземпляр симметричный ключ шифрования S_n может шифроваться с использованием асимметричного шифра с ключом U_W, чтобы формировать зашифрованный ключ K_n. Как E_n, так и K_n теперь могут сохраняться в мешке узла замка. Роль "только для записи" также может создавать ц-подпись с использованием ключа R_D и сохранять его. Аутентификация сообщений альтернативно или дополнительно может выполняться посредством надлежащего применения аутентифицирующей SDFT TAR-последовательности, которая может встраивать и автоматически складывать эту информацию для компактности и организационной простоты. Такие TAR-последовательности могут предоставлять возможность альтернативного способа аутентификации сообщений с использованием любых MAC-превращений по ключу. После того, как ролевой игрок "только для записи" может заканчивать запись, и экземпляр в оперативном запоминающем устройстве для S_n может быть уничтожен, ролевой игрок более может не иметь доступа к ключу S_n, чтобы дешифровать сообщение E_n, поскольку роль "только для записи" не может обладать асимметричного закрытого ключа R_w. Только те роли для доступа, которые могут обладать копией асимметричного закрытого ключа R_w, к примеру, роли читателя и писателя, могут дешифровать зашифрованный ключ K_n, чтобы получать S_n, и могут использовать его для того, чтобы управлять зашифрованным сообщением E_n, чтобы получать исходное сообщение T_n. Технология аутентификации дополнительно может включать в себя сцепление хешей или ц-подписей, аналогично способу, которым работают деревья Меркла, с возможностью задавать процесс аутентификации более эффективным для рабочих данных, содержащих множество отдельных сообщений. Ролевой доступ "только для записи" не может предотвращать неавторизованное усечение или перезапись предыдущих сообщений на узле замка, работающем посредством локальной системы; тем не менее, NUTS-экосистема может помогать предотвращать или подчеркивать такие ситуации посредством вовлечения признаков Nut-предыстории, репликации и синхронизации различными совместными способами. Это поясняется ниже в разделе относительно NutServer и модулей управления исправлениями.

Ограниченные характеристики ролей "только для записи" и "верификатора", представленных посредством таблицы на фиг. 87, могут помогать облегчать некоторые сложности, ассоциированные с широко распространенной загадкой "ключа Бога" в безопасности компьютерных систем. Они могут представлять собой известный класс проблем, при которых в одном случае системному администратору мо-

жет предоставляться "ключ Бога" или все учетные данные доступа к системе или к набору систем, чтобы поддерживать, модернизировать, восстанавливать, устанавливать и/или выполнять поиск и устранение неисправностей систем под рукой. Может быть предусмотрена такая тенденция в отрасли, чтобы автоматически коррелировать техническую способность с ускоренными проверками на отсутствие нарушения секретности вследствие относительно небольшого числа очень умелых и опытных системных администраторов с надлежащей проверкой на отсутствие нарушения секретности. Этот тип практики может сбоить при разрешении динамического характера доверчивых взаимосвязей, при котором уровень доверия между двумя сторонами может изменяться во времени односторонним способом, который может не быть обнаруживаемым посредством другого или может намеренно скрываться от другого. Посредством тщательного использования ролей для доступа "только для записи" и верификатора, рабочие данные могут защищаться от неавторизованного доступа в любой момент времени для транзитных или покоящихся данных. Применение этих двух ролей для доступа может обеспечивать возможность учреждению разделять соединенный характер технической способности и проверки на отсутствие нарушения секретности, чтобы полностью управлять каждым аспектом более надлежащим и независимым образом. Роль "только для записи" может разрешать пользователям и процессам вносить добавления в компонент журнала регистрации Nut-контейнера в качестве свидетельства обработки, но может не разрешать им читать рабочие данные или редактировать журнал регистрации. Дополнительно, роль "только для записи" может иметь доступ к обоим ключам ц-подписи и может создавать аутентификационные строки и верифицировать их. Роль верификатора может разрешать пользователям и процессам проверять Nut-контейнер на предмет внутренней согласованности и подлинности без предоставления доступа к рабочим данным. Узлы замка могут систематически модифицироваться, адаптироваться и вставляться в любой системе баз данных, такой как, но не только, noSQL или RDBMS, чтобы осуществлять такую степень детализации управления доступом на уровнях поля, записи, таблицы и/или базы данных. Компактность, гибкость, признаки и/или независимость могут обеспечивать возможность узлам замка существовать в компьютеризированных приборах в качестве встроенных шлюзов доступа к самому прибору. Это подробнее поясняется в последующем разделе относительно Интернета NUTS.

NAC-признаки могут охватывать полный набор перестановок для действий, которые могут предприниматься с целевыми рабочими данными. Простая матрица перекрестных ссылок разрешенных действий наряду с ее NAC-реализацией может быть показана следующим образом:

Действия	Чтение	Запись	Верификация
Чтение	Читатель	Писатель	Читатель
Запись	Писатель	Только для записи	Только для записи
Верификация	Читатель	Только для записи	Верификатор

Роли читателя и писателя могут иметь неявную способность верифицировать или аутентифицировать ц-подпись, содержащуюся в мешке узла замка.

Обобщим три способа защиты для узла замка: изменяемые замки, управление доступом к слою и/или управление Nut-доступом. Изменяемый замок может главным образом защищать мешок узла замка, который может использоваться для того, чтобы переносить некоторый контент данных. Управление доступом к слою может задавать то, как глубоко пользователь может проникать в слои графа замков. Управление Nut-доступом может указывать, какие части Nut-контейнера могут модифицироваться, просматриваться, записываться и снабжаться цифровой подписью пользователем. Некоторые или все эти уровни могут управляться посредством встроенных или сложенных наборов ключей в механизме замочных скважин узла замка. Механизм замочных скважин может представлять собой гибкий проход, который может предоставлять возможность вставки и обработки широкого спектра ключей шифрования для множества функций. Некоторые или все эти компоненты могут взаимодействовать и/или отдельно предлагать обширный набор средств управления доступом, которые могут индивидуально настраиваться в расчете на каждый Nut-контейнер и могут модульно конструироваться, чтобы демонстрировать поведение при замыкании, которое может требоваться для защиты контента. Модульность узла замка также может предоставлять простоту компоновки множества сложных замыкающих структур вследствие ее итеративного, компактного и модульного проектирования. Хотя множество различных алгоритмов могут использоваться для того, чтобы полностью отмыкать и использовать Nut-контейнер, информация, чтобы инициировать механизмы, может представляться посредством зашифрованных частей данных, которые могут сохраняться полностью в узлах замка Nut-контейнера, в силу чего его механизмы управления доступом могут быть портативными и могут перемещаться со своими рабочими данными независимо от внешних опорных мониторов. Эти механизмы дополнительно могут быть осуществлены посредством различных SDFT-способов и структур, чтобы помогать упрощать реализацию и лучше управлять сложностью внутреннего кодирования и/или подробностями данных.

Модели управления доступом Nut-контейнера могут представлять собой комбинацию (централизованного) мандатного управления доступом, (пользователь центрического) дискреционного управления

доступом и т.п. Они могут напоминать модель дискреционного управления доступом по способу, которым они могут сохранять некоторые или все атрибуты доступа внутри себя, и по методам, посредством которых владелец может непосредственно задавать уровни доступа в расчете на Nut-контейнер, чтобы упрощать транспортабельность. Они также могут приспосабливать некоторые или все модели мандатного управления доступом и могут интегрироваться в некоторые или все такие окружения вследствие своей гибкости, предоставленной посредством его замочных скважин, изменяемых замков и других механизмов. Кроме того, они могут демонстрировать другие характеристики, такие как, но не только, непрозрачность градиента, аддитивные атрибуты доступа и/или модульное связывание узлов замка, которые могут быть новыми для NUTS.

Обход узлов замка

Теперь можно обходить весь узел замка и видеть то, как вещи могут быть представлены, по ходу дела. Фиг. 88 иллюстрирует упрощенную схему, которая показывает потоки данных дешифрования в узле 8800 замка. Можно обратиться к элементам других чертежей, предусмотренных в этой переплетенной и интегрированной иллюстрации процесса отмыкания узлов замка, таких как фиг. 62, 78, 83, 84, 85 и 88. Можно обратиться к идентичной секции узла замка, пронумерованной посредством различных номеров элементов, но она может представлять другой вид идентичной секции, анализируемой в типе подхода на основе детализации. Упорядочение логических операций, которые могут требоваться в процессе отмыкания узлов замка, может быть дополнительно оптимизировано для эффективности и/или других целей. Процесс отмыкания узла замка, в силу этого в конечном счете и графа замков или Nut-контейнера, может предусматривать эти этапы, которые могут описываться в этом примере, такие как, но не только, использование первичных ключей, чтобы получать права доступа, и ключей дешифрования для узла замка, аутентификация узла замка, распространение прав доступа по всему графу замков, логическая операция изменяемого замка и/или дешифрование сохраненных данных; эти этапы могут расширяться, сужаться или переупорядочиваться по мере необходимости. При необходимости, определенные механизмы в графе замков и узле замка могут извлекать выгоду из соответствующего применения SDFT-способов.

Первичные ключи 8804 могут вставляться в секцию 8806 ввода, и каждый первичный ключ может использовать свой ассоциированный способ шифрования для того, чтобы пытаться дешифровать совпадающую зашифрованную карту 8810 ключей и развертывать ее в структуру карты 8808 ключей. Каждая карта 6240 ключей может формировать основной ключ 6210, который может использоваться посредством изменяемого замка 8812. В каждой карте 7840 ключей (эквивалентной 6240), может быть предусмотрен набор ключей 7850 слоя, и каждый ключ слоя (такой как 7852-7858) может вставляться в совпадающие узлы замка слоев (такие как 7820-7830) графа замков в соответствующей замочной скважине 8306 под первичный ключ секции 8302 ввода (в этом примере, ключ слоя, такой как 7852-7858, может быть эквивалентным первичному ключу в 8304); обозначенные посредством слоя узлы замка, такие как 7820-7830, могут использовать MAT-замок, который может требовать минимум двух ключей для того, чтобы открывать его: ключ слоя, к такой как 7852, и выведенный связующий ключ, такой как 8512, который может содержаться в секции 8510 или 8826 вывода. Для узла 8300 замка с замочной скважиной, в каждой карте 8310 ключей может быть предусмотрен набор ключей 8314 отмыкания наборов ключей атрибутов доступа (AAKSUK), называемых набором 8312 ключей доступа (AKS), и каждый AAKSUK-ключ может вставляться в замочную скважину 8320 под ключ доступа секции 8302 ввода текущего узла 8300 замка с замочной скважиной. После того, как набор ключей 8336 распространения атрибутов доступа (AAPK) может достигаться таким образом, они 8522 (эквивалентные 8336) могут вставляться в замочную скважину 8560 под ключ доступа следующего связанного узла 8540 замка. Теперь можно иметь набор 8332 ключей атрибутов доступа (AAKS), который может содержать ролевые ключи 8334 доступа (ARK). ARK может задавать роли для доступа первичного ключа 8304 для всего графа замков. Ц-подписи различных секций узла замка, таких как 8840-8848, могут аутентифицироваться с использованием этих ARK. Ц-подпись идентификатора замка и метаданных 8840 может аутентифицироваться с использованием открытого RAT ARK 8344 (он может представлять собой открытую часть пары асимметричных RAT-ключей, которая может описываться в технических требованиях NAC) и алгоритма аутентификации, указываемого в секции 8830. Для аутентификации, секция 8830 может отправляться в алгоритм аутентификации наряду с соответствующей ц-подписью 8840 и открытым RAT ARK 8344. Если аутентификация сбоит, в таком случае секция 8830 может быть скомпрометирована, и процесс отмыкания узлов замка может вызывать ошибку и может прекращать обработку. В случае успешной аутентификации, каждая ц-подпись зашифрованных карт 8842 ключей может аутентифицироваться для каждой зашифрованной карты ключей, соответствующей допустимому вставленному первичному ключу. Для аутентификации, каждая строка eKM 8810 может отправляться в алгоритм аутентификации наряду с соответствующей ц-подписью 8842 и открытым RAT ARK 8344. Если аутентификация сбоит, то eKM может быть скомпрометирован, и процесс отмыкания узлов замка может вызывать ошибку и может прекращать обработку. Если все соответствующие eKM успешно аутентифицированы, то каждая ц-подпись зашифрованного извлеченного ключа 8844 может аутентифицироваться. Для аутентификации, каждый eDK 8814 может отправляться в алгоритм аутентификации наряду с соответствующей ц-подписью 8844 и открытым RAT ARK 8344. Если аутентификация сбоит, то eDK может быть скомпрометирован, и про-

цесс отмыкания узлов замка может вызывать ошибку и может прекращать обработку. Если весь соответствующий eDK успешно аутентифицирован, то каждая ц-подпись зашифрованного набора 8846 ключей может аутентифицироваться. Для аутентификации, каждый eKS 8818 может отправляться в алгоритм аутентификации наряду с соответствующей ц-подписью 8846 и открытым RAT ARK 8344. Если аутентификация сбоит, то eKS может быть скомпрометирован, и процесс отмыкания узлов замка может вызывать ошибку и может прекращать обработку. Если весь соответствующий eKS успешно аутентифицирован, то каждая ц-подпись зашифрованного мешка 8848 может аутентифицироваться. Для аутентификации, каждый зашифр-мешок 8822 может отправляться в алгоритм аутентификации наряду с соответствующей ц-подписью 8848 и COR ARK 8348. Если аутентификация сбоит, то зашифр-мешок может быть скомпрометирован, и процесс отмыкания узлов замка может вызывать ошибку и может прекращать обработку. Если весь соответствующий зашифр-мешок успешно аутентифицирован, то этот узел замка может считаться полностью аутентифицированный. Следует отметить, что зашифр-мешок может аутентифицироваться с использованием ролевого ключа 8348 доступа класса читателей (COR). Это может быть справедливым для узлов замка, хранящих рабочие данные Nut-контейнера, но для узлов замка, хранящих Nut-метаданные в своих мешках, открытый RAT ARK вместо этого может использоваться для того, чтобы аутентифицировать их. После этого, на основе типа изменяемых замков, указываемого в секции 8830 параметров узла замка, соответствующий алгоритм 8812 изменяемых замков может предприниматься в каждой строке 8814 зашифрованного извлеченного ключа (eDK) с использованием набора основных ключей 7844 из карт 8808 ключей. Успешное отмыкание изменяемого замка 8812 посредством дешифрования eDK 8814 может приводить к одному или более извлеченных ключей 8816 (DK). Каждый извлеченный ключ может дешифровать соответствующую строку 8818 зашифрованного набора ключей (EKS), которая может сохраняться в параметрах 8802. Дешифрование eKS может формировать соответствующую структуру набора 8820 ключей, которая может хранить структуру секции 8826 вывода и ключ из мешка. Выведенный связующий ключ(и), который может содержаться в структуре набора 8820 ключей, может сохраняться в секции 8826 вывода, и он может функционировать в качестве ключа, который может вставляться в замочную скважину под первичный ключ связанного узла 8530 замка, если таковые имеются. Ключ из мешка может дешифровать строку 8822 зашифрованного мешка (зашифр-мешок), которая может сохраняться в секции параметров, с использованием соответствующего шифра. Дешифрованный мешок может хранить данные, такие как, но не только, рабочие данные Nut-контейнера (графа замков), метаданные относительно рабочих данных, метаданные Nut-контейнера, метаданные мешка, любая комбинация этих и/или других данных. Метаданные мешка могут указывать то, хранит мешок 8824 Nut-часть или рабочие Nut-данные. Если мешок хранит Nut-часть, он может указывать, какую Nut-часть он может представлять и другие соответствующие метаданные Nut-части и/или другие данные. Если мешок хранит рабочие данные Nut-контейнера, он может указывать то, могут сохраненные данные представлять собой фактические данные или ссылку на них, и если могут, то, какой тип ссылки они могут представлять собой, то, какую ссылку они могут представлять собой, и/или то, где они могут располагаться.

Эта последовательность этапов может повторяться для каждого узла замка в графе замков, чтобы отмыкать Nut-контейнер. Фиг. 89 показывает общую блок-схему последовательности операций способа Nut-отмыкания. Большинство этапов могут подробно описываться в предыдущем примере, но некоторым этапам может требоваться дополнительное разъяснение. Этап 8902: организация узлов замка в надлежащие уровни обхода: поскольку узлы замка могут сохраняться в строковой форме в структуре в виде списка, фактическая топология графа замков может извлекаться и конструироваться с использованием информации связывания, которая может сохраняться в каждом узле замка. После того, как граф может конструироваться, далее один или более дополнительных проходов могут осуществляться, чтобы надлежащим образом назначать уровни графа таким образом, что узлы замка могут обходиться в надлежащей последовательности. Этап 8908: предложение окна ввода для некоторых или всех замочных скважин на основе фразового пароля: во время обработки секции ввода, если замочная скважина на основе фразового пароля встречается с пустым ключом (фразовым паролем), то она может предлагать окно ввода фразового пароля. Это поведение по умолчанию может модифицироваться, чтобы вызывать другую функцию или обходить любые пустые замочные скважины под фразовый пароль. Любой логический этап или процесс на блок-схеме последовательности операций способа может иметь ошибки, которые могут вызывать и могут приводить к прекращению процесса, и они не указываются подробно, поскольку она представляет собой высокоуровневую блок-схему последовательности операций способа: например, любой процесс, который выполняет попытку операции, может сбоить и может прекращать алгоритм. Остальная часть блок-схемы последовательности операций способа может следовать вдоль тракта предыдущего примера.

Фиг. 90 иллюстрирует то, как система на основе NUTS может открывать документ, содержащийся в Nut-контейнере. Вводится прямая ссылка: NUTbook 9000 может представлять собой приложение для организации совокупностей данных, которое может использовать Nut-контейнеры и может по существу выступать в качестве персонального PKI когда дело доходит до сохранения и организации совокупностей паролей, ключей шифрования и/или сертификатов. Обозначения файлов, такие как 9002 и 9004, могут использоваться на всех схемах для того, чтобы представлять Nut-файл. В NUTbook-системе может

существовать Nut-контейнер 9002 основного NUTbook-ключа доступа, который может отмыкаться, чтобы получать некоторую минимальную функциональность из приложения. Ключ может сохраняться в Nut-контейнере 9002 и может называться основным NUTbook-ключом, и отмыкающий механизм в сам Nut-контейнер 9002 может содержать фразовый пароль. Может быть предусмотрена иерархическая взаимосвязь ключей с основным NUTbook-ключом 9002 и ключом 9004 доступа к документам таким образом, что для того, чтобы осуществлять доступ к любому документу, хранящему Nut-контейнеры в этой конфигурации, может требоваться ключ доступа к документам. Следовательно, иерархия может задаваться в качестве необходимости основному NUTbook-ключу 9002, чтобы открывать и осуществлять доступ к ключу 9004 доступа к документам. Nut-контейнер, хранящий ключ доступа к документам, может иметь Nut-идентификатор #33 9016. В силу этого, ключ, который может сохраняться в рабочих данных 9020, может упоминаться как идентификатор #33 ключа: как к ключу доступа к документам, так и к Nut-идентификатору Nut-контейнера, хранящего его, можно обращаться посредством идентичного идентификатора, в этом случае #33. Документ 9040 может сохраняться в Nut-контейнере 9030 с Nut-идентификатором #44 9036. Аналогично документ может упоминаться как документ #44. Когда пользователь решает открывать документ #44, одна из замочных скважин в замочной скважине 9032 под первичный ключ может указывать то, что ей может требоваться идентификатор #33 ключа, чтобы открывать его. Nut-контейнер #33 9004 может запрашиваться из NUTbook, и для того, чтобы его открывать, возможно, требуется открытие Nut-контейнера 9004. Для открытия этого Nut-контейнера, возможно, должен открываться Nut-контейнер 9002. Предположим, что пользователь может уже инициализировать свой NUTbook с фразовым паролем в Nut-контейнер 9002, и NUTbook может кэшировать основной NUTbook-ключ в запоминающем устройстве. Последующее открытие Nut-контейнера 9004 может требовать только дополнительного фразового пароля для доступа на уровне документов NUTbook, и как только он может открываться, может возникать каскад Nut-отмыканий, чтобы в конечном счете раскрывать дешифрованный документ #44 9050. NUTbook может кэшировать ключ доступа к документам на конечное количество времени, чтобы ускорять выборку документов в течение сеанса, но определенные события, такие как неактивность, спящий режим, экранный замок, тайм-ауты и/или явное замыкание, могут требовать ввода снова фразового пароля для доступа к документам. Этот раздел вводит концепты NUTbook-приложения и иерархических паролей, которые могут дополнительно поясняться в последующем разделе. Последовательность этапов, которые могут требоваться для того, чтобы открывать единый документ, может быть объемной, но часть или вся используемая логика могут быть основаны на узлах замка и их итеративных процессах, и большая их часть может скрываться от пользователя. Конечный результат может заключаться в том, что фрагмент данных может сохраняться в Nut-контейнере, к примеру, 9030, и его безопасность может быть согласованной в некоторых или всех окружениях.

Фиг. 91 иллюстрирует широкое использование в NUTS-диалекте, чтобы ссылаться на рабочие данные Nut-контейнера посредством Nut-идентификатора Nut-контейнера, хранящего его. Здесь, он показывает то, как замочная скважина 9124, тегированная для ключа #33, может фактически искать Nut-контейнер 9110 с Nut-идентификатором #33 9112, и он может ожидать, что Nut-контейнер #33 хранит один ключ 9114, который может вставляться в замочную скважину 9124. Может быть интересным отметить, что на множестве из этих схем и примеров, к имени файла Nut-файла, если Nut-контейнер может сохраняться в файле, можно редко обращаться в большинстве операций.

Следующий набор схем показывает различные примерные варианты осуществления графа замков, которые могут подчеркивать гибкость и выразительность модели на основе узлов замка и графа замков с использованием изменяемых замков и связывания узлов замка.

Фиг. 92 показывает упрощенный вариант осуществления модели замыкания по списку получателей: любой ключ 9210 может отмыкать узел 9220 замка "OR-замок", который может достигать узла замка, переносящего рабочие данные 9290 Nut-контейнера. Следует отметить, что для простоты, узел замка может графически представляться в качестве замка, но в реальности он представляет собой полностью функционирующий узел замка, который может сохранять некоторые метаданные для Nut-контейнера.

Фиг. 93 показывает упрощенный вариант осуществления модели упорядоченного замыкания: первым может представляться ключ 9310, затем вторым ключ 9320, который может предоставлять доступ к рабочим данным 9390 Nut-контейнера. Узел 9312 замка "MAT-замок" может требовать одного ключа, тогда как узел 9322 замка "MAT-замок" может требовать как ключа 9320, так и связующего ключа из узла 9312 замка.

Фиг. 94 показывает упрощенный вариант осуществления модели упорядоченного замыкания с главным ключом: первым может представляться ключ 9410, затем вторым ключ 9420, который может предоставлять доступ к рабочим данным 9490 Nut-контейнера. Альтернативно, главный ключ 9430 может представляться в узел 9432 замка "OR-замок" непосредственно, который может предоставлять доступ к рабочим данным 9490. Узел 9432 замка "OR-замок" может обеспечивать возможность связующему ключу или главному ключу отмыкать его.

Фиг. 95 показывает упрощенный вариант осуществления модели замыкания с главным ключом: ключ 9510 или главный ключ 9520 могут представляться вместе или отдельно, которые могут предоставлять доступ к рабочим данным 9590 Nut-контейнера.

Фиг. 96 показывает упрощенный вариант осуществления модели замыкания с главным ключом: ключ 9610 или главный ключ 9620 могут представляться вместе или отдельно, которые могут предоставлять доступ к рабочим данным 9690 Nut-контейнера. Размещение MAT-замка в графе 9632 замков может указывать то, что могут быть предусмотрены определенные элементы управления слоем для этого Nut-контейнера, и они могут представлять собой Nut-часть, сохраняющую некоторые Nut-метаданные.

Фиг. 97 показывает упрощенный вариант осуществления модели замыкания по принципу сейфовой ячейки: ключ 9710 и банковский ключ 9712 могут представляться вместе, которые могут предоставлять доступ к рабочим данным 9790 Nut-контейнера.

Фиг. 98 показывает упрощенный вариант осуществления модели замыкания по принципу разделения секретов с главным ключом: из набора ключей 9810, число ключей, удовлетворяющее или превышающее пороговое значение разделения секретов, может представляться вместе, которые могут предоставлять доступ к рабочим данным 9890 Nut-контейнера. Альтернативно, главный ключ 9820 может представляться в узел 9822 замка "OR-замок" непосредственно, который может предоставлять доступ к рабочим данным 9890. Ключи 9810 могут представлять собой любую комбинацию фразовых паролей, симметричных ключей и/или асимметричных ключей, поскольку структуры замочных скважин/карт ключей могут скрывать зубцы, которые могут требоваться посредством схемы разделения секретов, используемой в узле 9812 замка "SS-замок".

Фиг. 99 показывает упрощенный вариант осуществления модели замыкания по принципу PrivaTegrity: пользовательский ключ 9920 может представляться в OR-замок 9922, который может предоставлять доступ к рабочим данным 9990. Альтернативно, девять ключей 9910 могут представляться вместе в MAT-замок 9912, который может предоставлять доступ к рабочим данным 9990 Nut-контейнера. Модель по принципу PrivaTegrity предложена в начале 2016 года автором David Chaum, чтобы реализовывать систему обмена текстовыми сообщениями, которая может защищенно передавать сообщения с использованием ключей, известных ее пользователям, но она может иметь основанную на тайном сговоре систему черного хода, которая может предусматривать вплоть до девяти различных ключей, хранящихся посредством девяти международных юрисдикции, чтобы предоставлять доступ для правоприменения к конкретным сообщениям исключительно в том случае, если все девять юрисдикции могут соглашаться, что он является жизненно важным и может юридически гарантироваться.

Фиг. 100 показывает упрощенный вариант осуществления конфигурации с несколькими Nut-контейнерами, в которой несколько рабочих данных могут сохраняться в одном Nut-контейнере: пользовательские ключи 10010 или главный ключ 10012 могут осуществлять доступ к одним рабочим данным или к обоим из них, что может зависеть от их средств управления доступом к слою. Главный ключ 10020 может осуществлять доступ только к рабочим данным 10092 вследствие своего пути обхода через граф замков. Этот граф замков может отображать гибкость модульных узлов замка и ее уровней управления доступом, работающих вместе. Отдельные посылки данных могут защищаться различными способами частично или полностью в этом одном Nut-контейнере. Если главные ключи 10012 и 10020 могут быть идентичными, это может позволять держателя ключей осуществлять доступ к обоим рабочим данным.

Фиг. 101 показывает упрощенный вариант осуществления конфигурации с несколькими Nut-контейнерами: любой из пользовательских ключей 10110 может осуществлять доступ к некоторым или всем трем рабочим данным, что может зависеть от их средств управления доступом к слою. Ключи 10120 для SS-замка 10122 могут осуществлять доступ только к рабочим данным 10194 вследствие своего связывания узлов замка. Этот граф замков может отображать гибкость модульных узлов замка и ее уровней управления доступом, работающих вместе и/или отдельно. Отдельные посылки данных могут защищаться различными способами в этом одном Nut-контейнере. Гибкий характер этого раскрытия сущности может разрешать бесконечные варьирования конфигураций замыкания.

Фиг. 102 показывает упрощенный вариант осуществления модели прямого замыкания с несколькими рабочими данными: этот граф замков может показывать плоскую топологию для Nut-контейнера, а не обычную линейную. OR-замок 10212 может представлять собой интересный узел в том, что может быть предусмотрено несколько способов реализовывать несколько связующих ключей, требуемых для того, чтобы соединять его с пятью различными узлами замка. В одном варианте осуществления, секция вывода узла 10212 OR-замка может содержать пять выведенных ключей. В другом варианте осуществления карта выведенных связующих ключей может встраиваться в качестве карты ключей в замочную скважину и затем может распространяться в секцию вывода. Кроме того, ключи слоя также могут играть роль в отношении того, какие ключи могут осуществлять доступ к различным узлам.

Фиг. 103 показывает упрощенный вариант осуществления примера упорядоченной пересылки сообщений: он может представлять собой устойчивое к тайным сговорам проектное решение с использованием Nut-контейнеров и ключей на основе взаимосвязей (RBK, прямая ссылка). Mr. Data 10310 может иметь взаимосвязь с каждым из Alice, Bob, Charlie и Danny. Некоторые или все участники могут знать друг друга. Его взаимосвязь может быть преобразована в символьную форму в силу наличия ключей на основе взаимосвязей с каждым пользователем. Mr. Data может хотеть отправлять набор секретных инструкций каждому человеку, но он может хотеть, чтобы сообщения читались в определенной последовательности без возможности заглядывания вперед посредством тайного сговора между участниками. Сле-

довательно, Mr. Data может конструировать эти четыре Nut-контейнера с конкретным контентом в каждом из них. Nut-контейнер, отправленный Alice 10320, может открываться только посредством Alice, поскольку он может замыкаться с использованием RBK-набора между Alice и Mr. Data. Внутри 10320 может быть предусмотрено сообщение для Alice и ключ для Bob, K_{Bob}. Она может читать свое сообщение и может отправлять Bob ключ K_{Bob}. Nut-контейнер, отправленный Bob 10330, может использовать MAT-замок, который может открываться только с использованием двух ключей одновременно: RBK-ключа Bob между Bob и Mr. Data и ключа K_{Bob} от Alice. Внутри 10330 может быть предусмотрено сообщение для Bob и ключ для Charlie, $K_{charlie}$. Он может читать сообщение и может отправлять Charlie ключ $K_{charlie}$. Nut-контейнер, отправленный в Charlie 10340, может использовать MAT-замок, который может открываться только с использованием двух ключей одновременно: RBK-ключа Charlie между Charlie и Mr. Data и ключа $K_{charlie}$ от Bob. Внутри 10340 может быть предусмотрено сообщение для Charlie и ключ для Danny, K_{Danny}. Он может читать сообщение и может отправлять Danny ключ K_{Danny}. Nut-контейнер, отправленный в Danny 10350, может использовать MAT-замок, который может открываться только с использованием двух ключей одновременно: RBK-ключа Danny между Danny и Mr. Data и ключа K_{Danny} от Charlie. Внутри 10350 может быть предусмотрено сообщение для Danny. Он может читать сообщение, и план Mr. Data относительно упорядочения сообщений может сработать.

В области техники кибербезопасности, признак "черного хода" может ясно показывать отрицательные коннотации в различных диалогах, окружающих тему. Традиционно, механизмы черного хода могут реализовываться на прикладных уровнях, которые могут иметь разрешенный беспрепятственный доступ к данным, обрабатываемым посредством этого приложения. Этот тип доступа на прикладном уровне может истолковываться в качестве серьезной компрометации безопасности данных, обрабатываемых посредством этого приложения, в зависимости от того, какая сторона получает доступ к этому входу через черный ход. Восприятие компрометации в таких ситуациях может быть убедительным вследствие распространенности таких приложений, главным образом обрабатывающих незашифрованные данные в собственном запоминающем устройстве для хранения приложений, в силу этого потенциально предоставляя разрешение на доступ к открытым текстовым данным для пользователя черного хода. В NUTS и, в частности, в модели замыкания Nut-контейнера, некоторые могут рассматривать использование главного ключа в качестве типа черного хода в Nut-контейнер; тем не менее, технически это может очень отличаться, поскольку во всех моделях замыкания Nut-контейнера, все двери (замочные скважины) представляют собой парадные входы, и требует надлежащего криптографического ключа, чтобы получать доступ в Nut-контейнер. NUTS API или любой вариант осуществления на основе NUTS-связанных приложений может не иметь намеченного черного хода, спроектированного на прикладном уровне. Может быть множество законно серьезных оснований иметь записи главного ключа доступными для Nut-контейнеров, но все такие записи могут задаваться только посредством секретного ключа и могут быть непосредственно заметными посредством поверхностного анализа любой секции ввода узла замка. Следовательно, любое приложение, пытающееся устанавливать функциональность типа черного хода в NUTS-связанном приложении, может достигать этого только после первого получения доступа к главному ключу для целевого набора Nut-контейнеров, и она может быть применимой только к тем Nut-контейнерам, в которых этот главный ключ является допустимым. Это может иллюстрировать гибкость, разделение, защиту и/или способность к восстановлению после сбоев датацентрического подхода к безопасности Nut-контейнера.

В некоторых или всех способах управления доступом в NUTS, может быть предусмотрен шаблон скрытия криптографических ключей в инкапсулированных структурах данных, раскладывание которых может раскрывать другие ключи, которые могут предоставлять доступ к целевому набору данных. В вариантах осуществления, проиллюстрированных в этом раскрытии сущности, большинство этих способов скрытия ключей могут использовать способы инкапсуляции данных и/или складывания данных. Способ скрытия ключей доступа может представлять собой предпочтение, заданное посредством реализатора, либо он может представлять собой параметризованную настройку в каждом Nut-контейнере. Эти способы могут содержать складывание данных, инкапсуляцию данных, шифрование на основе атрибутов, функциональное шифрование, маркеры авторизации из опорных мониторов либо любой другой способ, который может предоставлять избирательное криптографическое раскрытие последующих ключей доступа при предоставлении материала для доступа, который дешифрует или отмыкает его криптографический механизм. Демонстративные варианты осуществления в этом раскрытии сущности могут выбираться за свою простую и прямую механику и свои известные характеристики. Другие эквивалентные механизмы могут упрощать или обеспечивать большую эффективность конкретных аспектов вариантов осуществления, но они по-прежнему могут по существу предоставлять идентичные функциональности, для управления доступом к атрибутам доступа, которые могут точно предоставлять разрешение на доступ к целевому набору данных и могут быть независимыми от опорных мониторов по умолчанию. Любая эквивалентная технология раскрытия атрибутов доступа может подставляться вместо способов, проиллюстрированных выше, чтобы предоставлять идентичный уровень защиты для контента Nut-контейнера.

На этом завершается раздел относительно Nut-контейнера и его внутренних операций. Внутренние механизмы могут быть осуществлены непосредственно или посредством использования SDFT-способов,

которые могут упрощать кодирование и управление таким вариантом осуществления. Рабочие данные Nut-контейнера могут представлять собой то, что в конечном счете может защищать Nut-контейнер, что может представлять собой любые допускающие хранение цифровые данные, такие как, но не только, текстовый файл, двоичное приложение, файл с изображениями, ключи доступа к удаленной системе, выполняемые сценарии, учетные данные для того, чтобы защищенно устанавливать соединение между компьютерами, полные базы данных, операционные системы, связи с другими Nut-контейнерами, потоковые данные и/или текстовые сообщения. Вследствие способности Nut-контейнера описывать то, что он может хранить, через свои обширные конфигурируемые метаданные, стандартный список общих типов файлов может быть далеким от его характеристик хранения. Архитектура на основе узлов замка может позволять рабочим данным охватывать Nut-контейнеры, в силу чего она может приводить к неограниченным логическим размерам контейнеров. Если полупроводниковые NUTS-совместимые микросхемы или схемы могут быть доступными, может быть возможным превращать физическое устройство в сам Nut-контейнер, в силу чего к устройству может осуществляться доступ только посредством держателя ключей. Последовательность таких устройств может составлять полные сети и сети intranet, которые могут работать только с надлежащей аутентификацией. Гибкий характер модульного проектирования на основе узлов замка может разрешать бесконечные варьирования конфигураций замыкания для Nut-контейнера. В следующих разделах, могут вводиться различные системы и/или способы, которые могут использовать Nut-контейнеры в качестве основы защищенной области хранения, чтобы показывать то, как некоторое общие услуги и технологии могут расширяться, улучшаться или перепроектироваться, чтобы предлагать возможности, которые могут казаться за пределами досягаемости среднестатистического пользователя.

Модульный ввод-вывод

Существенный объем усилий программиста может расходоваться на удостоверение в том, что данные могут надлежащим образом вноситься в программу, трансформироваться в ее пространство рабочего запоминающего устройства, вычисляться и/или редактироваться, а затем могут надлежащим образом постоянно сохраняться. Неприятный побочный продукт этого режима разработки приложений может представлять собой побочный продукт в виде окончательного устаревания форматов файлов и их различных версий. Владение, обладание и управление собственными данными могут быть полезными и желательными целями, но как их можно использовать, если невозможно их читать надлежащим образом? Способность читать формат, записывать формат, выполнять действие для читаемых данных и/или отображать читаемые данные может составлять некоторые фундаментальные компоненты типичной программы. Модульный ввод-вывод (MIO) может представлять собой систему и/или способ модуляризации этих логических операций в репозиторий модульных компонентов, которые могут использоваться любым, кто может осуществлять доступ к нему. Побочный продукт MIO может представлять собой способность создавать модули преобразования форматов файлов, которые могут обеспечивать возможность пользователям осуществлять доступ к предыдущим версиям процедур чтения и записи файлов таким образом, что их устаревшие данные могут быть читаемыми. Это может называться обратной совместимостью. Также может предлагаться концепт прямой совместимости, но полезность этого признака может зависеть от опыта программиста, который может проектировать модули приложений. Может быть предусмотрен предпочтительный вариант осуществления MIO-системы, в котором некоторые или все модули могут инкапсулироваться в Nut-контейнерах, в силу чего аутентификация, защита и/или управление доступом каждого модуля могут существовать по умолчанию. Фиг. 104 показывает типичные компоненты в модульном вводе-выводе. Элемент 10450 может представлять собой модульный репозиторий ввода-вывода (MIOR), который может представлять собой серверный процесс, который может сохранять и организовывать MIO-компоненты. MIOR может быть осуществлен в качестве локального и/или удаленного приложения серверного типа, которое может выступать в качестве интеллектуального кэша для таких модулей. В других вариантах осуществления, MIOR может иметь локальный кэш на локальном устройстве, так что это позволяет еще более упрощать обычно запрашиваемые MIO-модули. Типовое приложение 10402, которое может читать и/или записывать в файл 10404, может концептуально и программно разбиваться на модули, чтобы читать 10406 и записывать 10408 файл. Файл 10404 может форматироваться в конкретном формате A, который может быть конкретным для приложения 10402. Аналогично, этот чертеж показывают два других приложения 10410 и 10418 с соответствующими файлами 10412 и 10420 данных и их соответствующими модулями 10414, 10422, 10416 и 10424 чтения и записи, которые могут сохраняться в MIOR 10450 для конкретных форматов, которые могут представлять собой "B" и "C". MIOR может содержать другие модули, которые могут выполнять различные задачи или процедуры для приложения. Посредством 10426-10432 могут быть проиллюстрированы модули преобразования файлов, которые могут выполнять трансформации из одного формата файлов в другой, как указано посредством их соответствующих меток: модуль Convert_A_B 10426 может вовлекать данные, читаемые в запоминающее устройство приложения, из формата A файлов посредством модуля 10406 чтения файлов, и после этого он может трансформировать структуру запоминающего устройства в структуру, напоминающую структуру запоминающего устройства, которая может создаваться посредством модуля File_Read_B 10414 чтения файлов.

Модульный ввод-вывод: чтение и запись

Фиг. 105 показывает простую операцию чтения и записи с использованием MIOR 10500. Приложение 10502, которое может обрабатывать файлы, которые могут сохраняться в формате A файлов, может читать файл F_A 10504, отформатированный в формате A, посредством запроса модуля File_Read_A 10506 чтения файлов из MIOR 10500. Модуль 10506, если содержится, может передаваться 10510 в App_A 10502, причем в этот момент App_A 10502 может устанавливать и может выполнять модуль File_Read_A 10506 чтения файлов для файла F_A 10504. Модуль File Read_A 10506 может выполнять чтение файлов для файла F_A 10504 и может конструировать структуру внутреннего запоминающего устройства, которая может представлять контент файла F_A 10504. Эта структура запоминающего устройства, которая может представлять контент файла F_A 10504, затем может переноситься в вызывающее приложение App_A 10502. После успешного переноса, App_A 10502 может продолжать выполнять свои функции с контентом файла F_A 10504, который может присутствовать в его пространстве рабочего запоминающего устройства. В других вариантах осуществления, может не возникать необходимости в том, чтобы переносить структуру запоминающего устройства в App_A 10502, как только контент файла может читаться посредством модуля File_Read_A 10506 чтения файлов, если может быть средство, посредством которого как модуль 10506 чтения файлов, так и модуль 10502 приложений могут совместно использовать идентичное пространство в запоминающем устройстве.

Когда приложение App_A 10502 готово сохранять модифицированный контент файла F_A 10504 обратно в форму файла, оно может контактировать с MIOR и может запрашивать модуль записи файлов на предмет формата A файлов, называемого File_Write_A 10508. После приема 10512 модуля 10508, App_A может устанавливать и может выполнять его с использованием идентичной технологии для переноса структур запоминающего устройства для хранения приложений в качестве процесса чтения. Модуль 10508 записи может выполнять операцию записи в постоянное хранилище, которая может создавать модифицированный файл F_A 10520. Запросы в MIOR для модулей 10506 и 10508 чтения и записи могут осуществляться в любой последовательности, которая может считаться надлежащей разработчиком приложений. В одном варианте осуществления, приложение может запрашивать некоторые или все релевантные модули ввода-вывода заранее до продолжения, с тем чтобы быть уверенным в том, что некоторые или все необходимые операции ввода-вывода могут выполняться посредством приложения, что может предотвращать дальнейшие нежелательные сбои. В другом варианте осуществления может быть предусмотрен локально кэшированный MIOR ранее выбранных модулей посредством ранее запущенных приложений, который может поддерживаться, чтобы ускорять процедуры запроса и выборки.

Может быть предусмотрено множество способов переноса и/или совместного использования структуры запоминающего устройства между двумя или более логических процессов для специалистов в данной области техники, таких как, но не только, совместно используемые сегменты запоминающего устройства, файлы с распределением запоминающего устройства, базы данных, межпроцессные сообщения, двоичные файлы дампа запоминающего устройства и/или преобразованные дампы запоминающего устройства.

Предпочтительный способ переноса запоминающего устройства для хранения приложений в MIO-системе может заключаться в том, чтобы использовать преобразованные дампы запоминающего устройства между процессами. Функции JSON-чтения и записи могут модифицироваться, чтобы распознавать двоичные данные, и автоматически могут преобразовывать их в/из base64-кодирования или других схем кодирования двоичных данных в текст. Фиг. 106 показывает трансформации и переносы данных, которые могут быть предусмотрены в типичной операции чтения MIO-файлов. МЮ-модуль File_Read_A 10604 чтения может читать 10620 файл, называемый F_A 10602, в формате A в свое рабочее запоминающее устройство 10606. Таким образом, релевантный контент файла 10602 может представляться 10630 в структуре 10606 запоминающего устройства для хранения приложений. Запоминающее устройство для хранения приложений может сохраняться в JSON-совместимую структуру 10606 данных и может маршализоваться в текстовую форму 10610 с использованием вызова функции JSON-записи. Необязательно, JSON-вывод может встраиваться в Nut-контейнер 10608 для дополнительной защиты. Таким образом, запоминающее устройство 10606 для хранения приложений может преобразовываться и сохраняться 10608 за пределами модуля 10604 чтения. Nut-контейнер 10608 затем может открываться и читаться в запоминающее устройство посредством App_A 10612, и JSON-чтение может выполняться для посылки данных 10610, за счет этого реконструируя данные в рабочем запоминающем устройстве App_A 10614. Способы 10622 и 10624 переноса данных могут включать в себя, но не только, параметры командной строки, межпроцессные сообщения и/или файл(ы) данных. Приложение для чтения и/или приложение обработки данных могут представлять собой отдельные процессы на различных машинах, на идентичной машине, отдельные подпроцессы или отдельные ядра; или приложения могут представлять собой один логический процесс для локальной или удаленной машины с динамическими характеристиками, чтобы модифицировать его выполняемый код на лету.

Модульный ввод-вывод: обратная совместимость Приложения могут подвергаться прогрессивным изменениям во времени посредством выдачи изменений версии с улучшениями в течение его продолжительности действия. Множество этих изменений версии могут включать в себя изменения формата фай-

лов хранения, используемых для того, чтобы сохранять задания пользователя. Статистически, это может приводить к двум проблемам: обременение и устаревание. Обременение может возникать, когда программное обеспечение чрезмерно увеличивается по размеру вследствие добавления поддержки обратной совместимости в каждую версию для каждого изменения формата в течение срока службы линейки продуктов. Оно может предусматривать достаточное число изменений версии формата. Кроме того, если могут быть предусмотрены другие сторонние или открытые форматы, которые приложение может хотеть обрабатывать, то это может приводить к большему раздуванию программного обеспечения. Фиг. 105 иллюстрирует то, как для любой версии любого формата, который может использовать приложение, если модульные модули чтения и записи могут быть доступными в MIOR, то файл может читаться и обрабатываться без чрезмерного увеличения размера. Кроме того, фиг. 105 иллюстрирует то, как более новые модули чтения и записи могут независимо добавляться в MIOR, и каждое приложение, которое может обмениваться данными с MIOR, теперь может иметь доступ к дополнительным модулям форматирования без программных модификаций. Эти более новые модули могут иметь способность читать различные версии формата файлов для идентичной линейки продуктов приложения, или они могут представлять собой модули обеспечения совместимости, чтобы читать и записывать сторонние форматы файлов, написанные любым, в том числе и разработчиком приложений.

Фиг. 107 показывает пример обратной совместимости, в котором версия приложения App_B 10702 может быть более свежей, и может использовать соответствующую более новую версию B формата файла данных, но пользователь может хотеть читать и записывать устаревшую версию A формата 10704 файлов. Модули преобразования данных, такие как 10708 и 10710, могут создаваться и сохраняться в MIOR 10700 для таких случаев. Модуль преобразования может отвечать за чтение одного формата и формирование другого формата: в этом примере, модуль 10708 преобразования может читать файл формата A и может преобразовывать его в файл формата B; модуль 10710 преобразования может читать файл формата B и может преобразовывать его в файл формата A. Файл F_A 10704 может представляться в App_B 10702, при этом оно может определять то, что файл может иметь несовместимый формат, из его метаданных, и может продолжать выполнять запрос в MIOR 10700 на предмет последовательности модулей чтения, которые могут требоваться для того, чтобы читать A, и может формировать файлы B. MIOR 10700 может отвечать посредством отправки следующих модулей в App_B 10702: File_Read_A 10706, File_Write_A 10712, Convert_A_B 10708 и Convert_B_A 10710. App_B 10702 может активировать File_Read_A 10706 для файла F_A 10704, File_Read_A 10706 затем может активировать Convert_A_B 10708 и может переносить структуру запоминающего устройства в форме A в модуль 10708, затем модуль 10708 может преобразовывать принимаемые данные в форму B и может переносить их в App_B 10702. Когда App_B готово сохранять модифицированные данные обратно в файл F_A в формате A, то оно может активировать Convert_B_A 10710 и переносить структуру запоминающего устройства в форме B в модуль 10710, затем Convert_B_A 10710 может преобразовывать свою структуру запоминающего устройства в форму A и может активировать File_Write_A 10712 и может переносить структуру запоминающего устройства в форме A, затем File Write A 10712 может переписать файл F_A 10714 со своим модифицированным контентом в форме A и может форматировать записи файлов в формате A файлов. Более сложные примеры могут представлять собой примеры, в которых несколько модулей преобразования могут вызываться в последовательности, чтобы выполнять итеративный процесс преобразования в соответствующую устаревшую версию формата, либо разработчик может добавлять часто используемые модули преобразователя для изменения версии, такие как Convert B F и Convert_F_B, чтобы упрощать такие запросы.

Раздувание программного обеспечения может быть проиллюстрировано с помощью простого вычисления: предположим, что популярное приложение может подвергаться 5 крупным исправлениям, 3 версиям форматов файлов для 3 операционных систем с 3 крупными изменениями версии в каждой из них за 10 лет. Также предположим, что каждое из этих изменений может требовать различной версии процедур ввода-вывода для приложений. Это может потенциально приводить к самой актуальной версии приложения, которая переносит вплоть до 135 версий его функций ввода-вывода в себе. Если представить себе, что это может быть крайним случаем, можно понимать пролиферацию программного кода, который может формироваться с возможностью поддерживать обратную совместимость в приложении во времени. Эта характеристика может упоминаться как свойство обременения программного обеспечения.

Надлежащим образом поддерживаемый MIOR 10700 с согласованно обновленными модулями, добавляемыми в его репозиторий, может выступать в качестве библиотеки статистических форматов ввода-вывода и может обеспечивать возможность пользователям осуществлять доступ к устаревшим версиям их файлов данных в любое время в будущем: это позволяет разрешать проблемы устаревания программного обеспечения и формата данных. Когда приложение более не может создаваться, продаваться и/или обслуживаться, его срок эксплуатации может сокращаться радикально, поскольку более новые версии, которые могут обеспечивать возможность ему выполняться на более новых версиях операционной системы, не могут ожидаться. Когда приложение более не может выполняться на современных компьютерах вследствие несовместимостей, может затрудняться доступ к файлам данных, отформатированным по-

средством приложения. Умные пользователи и разработчики могут находить различные решения этих проблем, но это может требовать больших усилий и/или специализированных знаний с их стороны. Использование MIOR может требовать то, что по меньшей мере один разработчик должен поддерживать модули, которые могут быть ассоциированы с несуществующим на данный момент приложением, и он может задавать более новые версии модулей, которые должны периодически добавляться, которые могут быть совместимыми с более новыми версиями различных операционных систем. Этот тип регламентного обслуживания может быть автоматизирован с использованием инструментальных средств автоматизированного поблочного тестирования и соответствующих модулей автоматического формирования типа и версии ОС своевременно. Обновленные модули могут вставляться в MIOR, и все, кто может иметь доступ к MIOR, могут извлекать выгоду из работы разработчика; если конкретный MIOR может быть доступным для всех в Интернете, некоторые или все пользователи в Интернете могут извлекать выгоду из него автоматически без необходимости полной осведомленности пользователя в отношении низкоуровневых сложностей и тех процессов, которые могут активироваться, чтобы автоматически разрешать их. Сложности программной обратной и прямой совместимости могут упоминаться как свойство устаревания программного обеспечения.

Модульный ввод-вывод: прямая совместимость

Пользователь иногда может подвергаться ситуации, когда он мог покупать, устанавливать и/или использовать приложение много лет назад, но он не приобретал его последующие обновления за эти годы. Тем не менее, приложение по-прежнему может быть функциональным для него, но оно может только читать и записывать форматы файлов, которые могут быть совместимыми с его устаревшей версией приложения. Новейшая версия приложения может вводить более новый формат файлов с дополнительными признаками в какой-то момент в прошлом. Эта ситуация может представлять две проблемы для пользователя: 1) его версия приложения не может читать файлы, отформатированные в последней версии формата, и 2) другие программы, которые могут читать последний формат из этого приложения, могут не иметь возможность осуществлять доступ к его устаревшим отформатированным данным. Решение первой проблемы может называться операцией чтения для обеспечения прямой совместимости, за счет которой его устаревшее приложение может непосредственно загружать набор модулей из MIOR, которые могут выполнять прогрессивные преобразования для данных, которые могут обеспечивать ему возможность читать файлы, отформатированные в более новой версии, с использованием своей устаревшей программы. Решение второй проблемы может называться операцией записи для обеспечения прямой совместимости, за счет которой его устаревшее приложение может непосредственно загружать набор модулей из MIOR, которые могут выполнять прогрессивные преобразования для данных, которые могут обеспечивать ему возможность записывать файлы, отформатированные в более новой версии, с использованием своей устаревшей программы. Программы, компонуемые с прямой совместимостью в памяти, могут упрощать и повышать прозрачность этого типа перехода с использованием MIOR с минимальными потерями или вообще без потерь функциональности. Более новые признаки, предлагаемые в более свежих версиях формата, могут оптимально преобразовываться в менее сложные конструкты приложения или могут просто заменяться на необработанные данные и обеспечивать возможность пользователю модифицировать их позднее. Фиг. 108 иллюстрирует эти две различных логических операции с примерами.

Операция чтения для обеспечения прямой совместимости: App A 10802 может быть совместимым с файлами, отформатированными в версии A, но пользователь может хотеть читать более новый формат C файлов. Этот запрос может передаваться в MIOR 10800, и он может отвечать с последовательностью модулей, которые могут выполнять эти регрессивные преобразования: File_Read_C 10806, Convert_C_B 10808 и Convert_B_A 10810. Модуль File_Read_C 10806 может читать файл F_C 10804, который может форматироваться в версии C. Модуль 10806 затем может активировать функцию Convert C B 10808 регрессивного преобразования и может переносить структуру запоминающего устройства в нее. Модуль Convert C B 10808 может выполнять преобразование для данных в запоминающем устройстве и может формировать структуру запоминающего устройства, совместимую с форматом B, предыдущей версией формата файлов приложения. Модуль 10808 затем может активировать функцию Convert_B_A 10810 регрессивного преобразования и может переносить структуру запоминающего устройства в нее. Модуль Convert_B_A 10810 может выполнять преобразование для данных в запоминающем устройстве и может формировать структуру запоминающего устройства, совместимую с форматом A, требуемую версию формата файлов, совместимую с устаревшим приложением App A. Модуль 10810 может переносить структуру запоминающего устройства в формате A в вызывающее приложение App_A 10802, и App A может обрабатывать его. Таким образом, более новая версия формата файлов может читаться посредством устаревшей версии приложения без модификаций приложения.

Операция записи для обеспечения прямой совместимости: App A 10840 может быть совместимым с файлами, отформатированными в версии A, но пользователь может хотеть записывать более новый формат C файлов, который может находиться за рамками его исходных характеристик. Этот запрос может передаваться в MIOR 10800, и он может отвечать с последовательностью модулей, которые могут выполнять эти прогрессивные преобразования: File Write C 10816, Convert B C 10814 и Convert_A_B 10812.

App_A 10840 может активировать Convert_A_B 10812 и может переносить структуру запоминающего устройства в него. Модуль Convert A B 10812 может выполнять преобразование для данных в запоминающем устройстве и может формировать структуру запоминающего устройства, совместимую с форматом B. Модуль 10812 затем может активировать функцию Convert B C 10814 прогрессивного преобразования и может переносить структуру запоминающего устройства в нее. Модуль Convert B C 10814 может выполнять преобразование для данных в запоминающем устройстве и может формировать структуру запоминающего устройства, совместимую с форматом C. Модуль 10814 затем может активировать функцию File_Write_C 10816 записи файлов и может переносить структуру запоминающего устройства в нее. Модуль File_Write_C 10816 может записывать файл F_C 10818, который может форматироваться в версии C, в требуемой версии формата файлов. Таким образом, более новая версия формата файлов может записываться посредством устаревшей версии приложения без модификаций приложения.

Это раскрытие сущности не ограничено посредством этих двух показанных примеров. Модули преобразования могут формироваться с возможностью осуществлять доступ к некоторым или всем версиям форматов файлов для приложения в любой операционной системе. Модули преобразования могут не быть ограничены преобразованиями в рамках своей линейки продуктов приложения, а могут быть написаны с возможностью выполнять преобразования между различными линейками продуктов приложения. Модули преобразований могут включать в себя преобразования данных в различные форматы, такие как, но не только, файл в базу данных, база данных в файл, файл в поток данных, поток данных в файл, файл в веб-страницу, веб-страница в файл, файл в облачное хранилище данных, облачное хранилище данных в файл и/или другие.

Модульный ввод-вывод: отображение

Фиг. 109 показывает схему MIOR-модуля отображения при работе. После того, как приложение App A 10902 может успешно читать в свое запоминающее устройство данные из файла F_A 10904, он может переходить к использованию функциональности модуля Display A 10908 для того, чтобы отображать контент файла F A 10904 пользователю. В модулях отображения, функциональный аспект модуля может значительно варьироваться в зависимости от приложения, контента данных и/или проектного решения разработчика. Некоторые модули могут проектироваться с возможностью использовать способы на основе совместно используемого запоминающего устройства, которые могут обеспечивать возможность модулю отображения непосредственно осуществлять доступ к данным в запоминающем устройстве для хранения приложений, другие могут переносить данные в модуль отображения, который позволяет показывать их. Другие варианты осуществления модуля отображения могут представлять собой инструкции и/или шаблоны форматирования экрана для типа данных, которые должны быть показаны и, возможно, могут редактироваться. Эта модуляризация функциональностей отображения может предоставлять возможность создания настраиваемых модулей отображения для различных аппаратных и ОС-платформ при предоставлении возможности вызывающей программе оставаться относительно неизменной.

Архитектура на основе каталога совокупностей, поясненная ниже в разделе относительно NUTbook, может использовать простой аспект модульного отображения. Вместо компоновки еще более крупных монолитных приложений, чтобы обрабатывать, отображать и/или редактировать различные совокупности наборов данных, NUTbook может широко использовать MIOR-архитектуру, которая может обеспечивать возможность его индивидуальных настроек по частям на основе типа рабочих данных в анализируемом Nut-контейнере.

Модульный ввод-вывод: приложение

На фиг. 110 MIOR 11000 может сохранять модули модульных приложений, такие как 11022. NUTbrowser 11020 (прямая ссылка) может представлять собой приложение, которое может быть аналогичным по виду и поведению большинству браузеров файлов и каталогов, но которое может распознавать Nut-контейнеры и может реагировать на них посредством учета обширных метаданных Nut-контейнера. Внутри метаданных 11002 Nut-контейнера 11030, может быть предусмотрена информация, связанная с типом рабочих данных, которые он может защищать и сохранять. Когда пользователь выбирает Nut-контейнер из NUTbrowser 11020 и дважды щелкает, чтобы открывать его, NUTbrowser может открывать Nut-контейнер и может читать метаданные, чтобы выяснять, какие модули могут требоваться для того, чтобы открывать файл. Метаданные могут включать в себя данные, такие как, но не только, версия приложения, версия формата файлов и/или версия отображения. Затем NUTbrowser может выполнять запрос 11004 в MIOR 11000 на поиск приложения App_A 11022, File_Read_A 11024 и Display_A 11026. MIOR 11000 может возвращать некоторые или все модули, и приложение App_A 11022 может активироваться посредством NUTbrowser 11020. После того, как App A выполняется, оно может активировать File_Read_A 11024, чтобы читать контент рабочих Nut-данных F A 11028, которые могут сохраняться в Nut-контейнере 11030. После переноса структуры запоминающего устройства из 11024 в вызывающий модуль App A 11022, он может активировать модуль Display_A 11026 отображения, чтобы показывать данные F_A 11028 пользователю.

Модули приложений модульного ввода-вывода могут значительно варьироваться в том, что они могут хранить и осуществлять: в некоторых вариантах осуществления, они могут представлять собой

сложный логический вычислительный модуль; в другом варианте осуществления, они могут сохранять весь установочный комплект программного обеспечения; в другом варианте осуществления, они могут содержать некоторые или все аспекты функций ввода-вывода, отображения и/или приложения; в другом варианте осуществления, они могут содержать информацию, содержащую образующий Nut-контейнер, который может запускать "с ходу" перевоплощение окружения пользователя удаленным способом. Функциональность модулей приложений модульного ввода-вывода не ограничена этими случаями.

Признаки модульного ввода-вывода, такие как чтение, запись, отображение и/или приложение, могут перекрываться с механизмами управления доступом на уровне MIOR или контейнеров таким образом, что только надлежащим образом авторизованные пользователи могут осуществлять доступ к ним. Эти механизмы управления доступом могут включать в себя, но только, политики управления доступом, требования по владения и/или DRM-механизмы в ренумеративных целях. Большая часть средств управления доступом может вытекать из свойств Nut-контейнеров, в которых могут сохраняться модули. При более подробном пояснении этого раскрытия сущности далее, предоставляется прояснение в отношении механизмов, посредством которых могут извлекаться эти MIOR-запросы. Когда файл данных или его контент может инкапсулироваться в защищенном Nut-контейнере, может быть предусмотрено множество уровней метаданных, доступных относительно контента Nut-контейнера, эти метаданные могут указывать подробности формата данных, такие как, но не только, версия приложения, которая создает его, версия отображения, версия формата файлов, размер, время создания, время последней модификации, автор, тип файла и/или сводки. Атрибуты окружения, такие как, но не только, версия ОС, версия приложения, аппаратная модель и/или версия, могут предоставляться посредством приложения, которое открывает Nut-контейнер. При помощи этих фрагментов информации относительно окружения, контента данных и/или запрашиваемой операции, MIOR может искать надлежащие модули и может отвечать обратно либо с набором модулей, чтобы удовлетворять операцию, либо с сообщением об ошибке. Эти модули модульного ввода-вывода могут выполняться в качестве одиночных или отдельных процессов на идентичной машине, на различных машинах, на различных микросхемах или ядрах, по сети и в других режимах выполнения логического процесса(ов) для вычислительного устройства. Через эти модули, проблемы устаревания, обременения, адаптируемости, совместимости и/или гибкости могут разрешаться частично или полностью.

NUT-предыстория Nut-контейнер может быть структурирован с возможностью сохранять предысторию рабочих данных. Форма предыстории может содержать периодические снимки экрана, прогрессивные дельты, полные последовательности событий или любую комбинацию этих трех или любых других способов архивации. Форма предыстории может варьироваться в зависимости от типа сохраняемых данных и предпочтений и проектирования приложения и/или данных. NUTS-экосистема может включать в себя способы и системы, чтобы поддерживать эти режимы архивации предыстории данных. Три способа архивации могут представлять собой общепринятые способы, известные специалистам в данной области техники. Физическое местоположение Nut-предыстории может быть в Nut-части, называемой хвостом (фиг. 81), и ее непрозрачность может управляться посредством RAT Nut-контейнера.

Фиг. 111 показывает упрощенную Nut-семантику, которая иллюстрирует прогрессивные изменения в структуре предыстории по трем точкам во времени, охватывающие два сеанса редактирования документа. Во время t1, Nut-контейнер 11102 может хранить данные D1 11110, и его предыстория может быть пустой 11112. Пользователь может редактировать 11126 данные D1 11110 во время T2 и может формировать новую версию D2 11114. Приложение может использовать способ архивации снимков экрана и может сохранять исходную версию данных D1 11110 в качестве снимка экрана данных во время T1 11118 в секции 11116 предыстории Nut-контейнера. Затем, пользователь может редактировать 11128 данные D2 11114 во время T3 и может формировать новую версию D3 11120. Приложение может использовать способ архивации снимков экрана и может сохранять устаревшую версию данных D2 11114 в качестве снимка экрана данных во время T2 11124 в секции 11122 предыстории Nut-контейнера. Во время t3, секция 11122 предыстории теперь может хранить два различных снимка 11118 и 11124 экрана предыдущих версий данных D3 11120. Предыстория 11122 Nut-контейнера может просматриваться и извлекаться пользователем по желанию с использованием простых способов извлечения предыстории в любое время, что обеспечивает возможность реверсий или создания полностью новых документов из них. Могут быть предусмотрены параметры Nut-метаданных, которые могут управлять типом, частотой и/или долговечностью секции предыстории, чтобы задавать обоснованный рост предыстории для данных под рукой. Для некоторых текстовых документов, практично сохранять некоторые или все изменения навсегда, поскольку их размер может быть относительно небольшим при использовании дельта-способа архивации. Это может обеспечивать возможность Nut-контейнеру формировать некоторые или все сохраненные версии документа в любое время после этого. Двоичные документы могут архивироваться в качестве снимков экрана или дельт в зависимости от приложения. Определенные управляемые событиями приложения могут архивировать полный набор последовательностей событий в качестве своей предыстории. Следует отметить, что Nut-предыстория может архивироваться в самом Nut-контейнере и может быть независимой от внешних программ и/или систем хранения данных. При условии, что может быть предусмотрен способ архивации, доступный для типа рабочих данных в NUTS-окружении, любые рабочие

данные могут статистически архивироваться таким способом.

NUT-журнал регистрации Nut-контейнер может быть структурирован с возможностью сохранять журнал событий Nut-контейнера. Поскольку компьютерные процессы могут читать, манипулировать и/или записывать Nut-контейнер, они могут формировать и оставлять журнал аудита логических операций, выполняемых в Nut-контейнер, в самом Nut-контейнере. Журнал аудита по существу может существовать в расчете на каждый объект с точки зрения объекта. Следовательно, между Nut-предысторией и Nut-журналом регистрации, хроникой событий с начала для объекта данных может сохраняться в одном контейнере для дальнейшего рассмотрения позднее. Точность, контент и/или степень детализации архивов Nut-контейнера может зависеть от дисциплинированного и методического использования этих признаков разработчиками приложений, которые работают с Nut-контейнерами. Физическое местоположение Nut-журнала регистрации может быть в Nut-части, называемой витой (фиг. 81), и его непрозрачность может управляться посредством RAT Nut-контейнера.

Фиг. 112 показывает упрощенную Nut-семантику, которая иллюстрирует прогрессивные изменения в структуре журнала событий по трем точкам во времени, охватывающие два события, возникающие в Nut-контейнере. Этот пример может продолжать сценарий из фиг. 111 для Nut-предыстории. Во время t1, Nut-контейнер 11102 может хранить данные D1 11110, и его журнал 11212 регистрации может хранить одну запись 11218 в журнале регистрации для события E1, которая может указывать то, что Nut-контейнер 11102 создан во время T1. Пользователь может редактировать 11226 данные D1 во время T2, которые могут создавать новую версию данных D2 11114 в Nut-контейнере 11104. Приложение редактирования может регистрировать запись в журнале событий в T2 в Nut-журнал 11216 регистрации, который может указываться посредством элемента 11222. Затем, пользователь может редактировать 11228 данные D2 11114 во время T3 и может формировать новую версию D3 11120. Приложение редактирования может регистрировать запись в журнале событий в T3 в Nut-журнал 11230 регистрации, который может указываться посредством элемента 11224. Во время t3, секция 11230 журнала регистрации теперь может хранить три различных записи 11218, 11222 и 11224 в журнале событий. Журнал 11230 регистрации Nut-контейнера может просматриваться и извлекаться пользователем по желанию с использованием простых способов извлечения журнала регистрации в любое время, которые могут обеспечивать возможность аудитов для Nut-контейнера. Могут быть предусмотрены параметры Nut-метаданных для того, чтобы управлять типом, частотой и/или долговечностью секции журнала регистрации, чтобы задавать обоснованный и соответствующий рост размера журнала регистрации для Nut-контейнера.

Системные администраторы и разработчики приложений могут знать действия и усилия, которые могут быть предусмотрены при выслеживании дефектов и ошибок на своих системах, когда более одного приложения может вовлекаться в модификацию объекта данных, поскольку они, возможно, должны отсматривать журналы событий некоторых или всех способствующих приложений (если они могут иметь доступ к ним вообще) и могут отфильтровывать те записи в журнале событий, которые связаны с рассматриваемым объектом, а затем, возможно, вручную реконструировать события в последовательности, в которой они могут возникать в объекте. При использовании Nut-журнала регистрации, этот сбор журналов событий, фильтрация и реконструирование могут уже осуществляться на уровне объекта с точки зрения объекта. Кроме того, метаданные Nut-контейнера могут указывать в рабочее приложение уровень детализации подробностей сообщения журнала событий, которые могут требоваться владельцем объекта. Эта степень детализации может варьироваться от кратких до подробных уровней отладки, чтобы разыскивать различные строки запросов. Чувствительные, совершенно секретные рабочие данные могут требовать наиболее детализированного уровня подробностей журнала событий, чтобы выполнять журнал аудита касательно предыстории доступа. Вкратце, он может представлять собой согласованный и индивидуально настраиваемый способ управления подвергаемым аудиту прошлым объекта посредством любого приложения в расчете на каждый объект согласно уровню детализации, запрошенному посредством упомянутого объекта. Согласованный термин может означать согласованное проектное решение и операции доступного признака регистрации, и термин "индивидуально настраиваемый" может означать предпочтения в расчете на объект, которые может приспосабливать проектное решение.

Ключи на основе взаимосвязей (RBK).

Описание того, как ключи на основе взаимосвязей (RBK) могут устанавливаться, должно звучать знакомо всем, кто может использовать инструментальные средства шифрования вручную: Bob и Alice могут хотеть обмениваться данными в закрытом порядке и в силу этого они могут торговать случайно сформированными асимметричными ключами шифрования (только открытыми частями) между собой и могут использовать их в таком инструментальном средстве, как PGP или его эквивалент, чтобы обмениваться зашифрованными сообщениями и документами. Защита и управление парами ключей посредством Bob и Alice может оставляться исключительно на них. Это может иметь тенденцию быть преднамеренной и трудоемкой задачей для установления, поддержания и использования каждой взаимосвязи надлежащим образом, что, возможно, требует от Alice и Bob иметь краткую информацию и т.п. относительно шифров, их надлежащего использования и/или защиты ключей. Этот тип обмена ключами может возникать, когда Bob или Alice не имеет установленного сертификата открытых ключей через централизованный каталог или веб-сеть доверия. Это также может происходить, если любой участник чувствует,

что добавленный слой конфиденциальности может требоваться посредством создания полностью закрытого канала связи.

Что может происходить, если RBK представляет собой способ по умолчанию связи для людей, таких как Alice и Bob? Какие могут быть последствия, и что может требоваться для того, чтобы это происходило безболезненным способом? Систематические аспекты установления, обслуживания и/или использования RBK могут быть автоматизированы. Может быть конструктивным изучать некоторые свойства и последствия согласованного применения RBK до тщательного исследования подробностей того, как это может осуществляться систематически.

Характеристики ключей на основе взаимосвязей

Уровень доверия между двумя сторонами может представлять собой динамический регулируемый параметр. Он может представлять собой наблюдение за реальными взаимосвязями между любыми двумя сторонами: доверие может быть относительным. Оно может увеличиваться и уменьшаться во времени на основе событий и связи.

Одностороннее регулирование уровней доверия. Любая из сторон во взаимосвязи может в одностороннем порядке изменять уровень доверия взаимосвязи по желанию с/без информирования другой стороны.

Работоспособность канала взаимосвязи может определяться из контекста сообщения. Системы и ключи могут время от времени быть скомпрометированы для любого. Использование по умолчанию RBK может обеспечивать возможность любой из сторон анализировать контент связи и может определять вероятность компрометации систем или ключей другого пользователя. В простейшем случае, сообщение, исходящее от Bob без RBK-шифрования, может представлять собой знак компрометации.

Истинный характер взаимосвязи может оцениваться во времени. Если сообщение необычного характера передается через RBK, и ключ отправляющей стороны не скомпрометирован, то отправляющая сторона может изменять характер взаимосвязи.

Потеря взаимосвязи может быть безвозвратной, и часть или вся предыстория взаимосвязи может терять коммерческую и/или значимую ценность. В одностороннем порядке, любая из сторон может разъединять взаимосвязь посредством блокирования ее сообщений или стирания их RBK-набора. Эта логическая операция канала взаимосвязи может представлять для каждого пользователя детерминированное способность односторонней блокировки сообщений.

Стороны могут строго соблюдать взаимно применимые основополагающие правила или рисковать терять "взаимосвязь/основополагающие правила", которые могут варьироваться во времени. Нарушение неявных основополагающих правил может приводить к одностороннему разъединению взаимосвязи на безвозвратной основе, в цифровом смысле.

Это может обеспечивать возможность более тесного выражения взаимосвязей реального мира в цифровой криптографической форме. Криптография с открытым ключом в своей наиболее широко используемой форме может представлять собой централизованную модель, которая может противоречить тому, как люди формируют взаимосвязи. RBK могут быть децентрализованными и могут использовать криптографию с открытым ключом закрытым способом.

Изоляция расшатывания. Расшатывание RBK в окружении Bob может изолироваться до Bob и RBK-каналов, которые он может устанавливать со своими контактами, т.е. Alice. Ущерб окружения Alice может изолироваться до ее канала с Bob и их взаимных статистических коммюнике. Некоторые или все другие каналы взаимосвязи для Alice могут быть защищенными и не могут нарушаться хакерами, которые расшатывают окружение Bob.

Персональный информационный менеджер или PIM может представлять собой известный концепт приложения в компьютерном программном обеспечении. Он может широко задаваться в качестве амальгамы различных функций, которые могут предоставлять рабочие и организационные инструментальные средства для использования человеком. PIM может предлагать такие инструментальные средства, как, но не только, календарь, адресная книга, управление контактами, хранитель паролей, заметки, почтовый диспетчер, функция чатов, управление проектами, диспетчер ключей, калькулятор, списки задач и/или регистратор активности. PIM может представлять собой комбинацию любой из этих функций, либо она может просто предлагать одну функцию. PIM может проектироваться с возможностью работать локально изолированным способом либо только на PIM-веб-сервере, либо в любой комбинации вышеозначенного. В дальнейших пояснениях, ссылки на такие функциональности PIM, как адресная книга или чат, или почтовый диспетчер, либо могут пониматься как PIM, который предлагает любую из этих функций в качестве части своих предложений, или они могут представлять собой исключительную функцию.

Фиг. 113 показывает то, как запись цифровой адресной книги между Alice и Bob может быть структурирована, чтобы поддерживать RBK согласованным способом для некоторых или всех взаимосвязей в адресной книге. Адресная книга 11310 Alice, которая может представлять собой функцию, предлагаемую посредством ее PIM, может иметь две записи: запись 11320 для себя и запись 11330 для информации Bob. В собственной записи 11320 Alice, она может перечислять собственные базовые контактные данные 11322 и некоторые персональные данные 11324.

Запись 11320 адресной книги Alice может сохраняться и может защищаться в качестве рабочих

данных в Nut-файле в системе Alice. В контактной карте 11330 Bob, Alice может иметь некоторую контактную информацию для Bob 11332, такую как его имя и адрес электронной почты. Запись 11330 адресной книги Bob может сохраняться и может защищаться в качестве рабочих данных в Nut-файле в системе Alice. Адресная книга 11350 Bob, которая может представлять собой функцию, предлагаемую посредством его PIM, может иметь две записи: запись 11360 для себя и запись 11370 для информации Alice. В собственной записи 11360 Bob, он может перечислять собственные базовые контактные данные 11352 и некоторые персональные данные 11354. Запись 11360 адресной книги Bob может сохраняться и может защищаться в качестве рабочих данных в Nut-файле в системе Bob. В контактной карте 11370 Alice, Bob может иметь некоторую контактную информацию для Alice 11372, такую как ее имя и адрес электронной почты. Запись 11370 адресной книги Alice может сохраняться и может защищаться в качестве рабочих данных в Nut-файле в системе Bob. Когда Alice и Bob решают устанавливать RBK друг с другом, они могут решать устанавливать закрытый канал двунаправленной связи между собой. Alice может начинать процесс посредством формирования пары 11334 и 11336 асимметричных ключей, сохранения их в адресной карте 11330 Bob и передачи 11344 открытой части ключа 11334 для Bob. Процесс 11344 передачи может осуществляться посредством защищенного фразовым паролем Nut-контейнера, сообщения, записанного на бумаге, телефонного вызова с Bob, сообщения с использованием открытого ключа Bob, известного миру, или любой версии протоколов защищенного обмена ключами, известных специалистам в данной области техники. Когда Bob принимает это сообщение с ключом 11334 внутри, он может сохранять его в записи 11370 в адресной карте Alice в качестве ключа 11334 для отправки сообщений Alice в закрытом порядке. Bob затем может формировать пару 11374 и 11376 асимметричных ключей, сохранять их в адресной карте 11370 Alice и передавать 11346 открытую часть ключа 11374 Alice с использованием открытого ключа, который Alice отправляет 11334 ему, чтобы шифровать сообщение. Когда Alice принимает это сообщение, она может дешифровать 11336 сообщение от Bob с использованием своего закрытого ключа для сообщений Bob. Она может извлекать ключ 11374 внутри, она может сохранять его в записи 11330 в адресной карте Bob в качестве ключа 11374 для отправки сообщений для Bob в закрытом порядке. Она может создавать подтверждающее сообщение для Bob, зашифрованное с помощью ключа 11374 из карты 11330, и может отправлять его для Bob через любую рабочую среду связи. Bob может принимать сообщение, затем он может дешифровать его с ключом 11376 из карты 11370 и может помечать свой RBK-набор, который должен устанавливаться и быть активным с Alice.

Этапы для этого RBK-установления между Alice и Bob могут быть автоматизированы и могут инициироваться с помощью одной кнопки действий или команды. Это может представлять собой функциональную основу того, как NUTbook управляет своей совокупностью контактов, и может поясняться в разделе относительно NUTbook ниже в этом документе. Процесс может повторяться посредством Bob или посредством Alice независимо для некоторых или всех контактных карт в их соответствующих адресных книгах в их PIM. В конечном счете, каждый пользователь может устанавливать RBK-канал для каждого из своих контактов, которые могут рассматриваться в качестве закрытых каналов связи для каждой из их взаимосвязей. Если Cathy является общей подругой Alice и Bob, RBK-взаимосвязь Cathy с Bob может отличаться от RBK-взаимосвязи Cathy с Alice, и RBK-конфигурация может отражать эту реальность.

Теперь, когда, возможно, задан RBK и контекст его систематического использования, что он может сделать для Alice или Bob? Согласованное использование RBK для того, чтобы отправлять сообщения между двумя объектами, может предоставлять возможность мониторинга работоспособности их канала связи. Пример практического применения может заключаться в уменьшении числа спамовых почтовых сообщений. Может оцениваться то, что значительный объем глобальной полосы пропускания Интернета и хранения данных может заниматься посредством спамовых почтовых сообщений и посредством злоумышленных и коммерческих типов сообщений. Можно рискнуть предположить, что немногие люди могут приветствовать такие объемы спама. Некоторые обычные способы уменьшения спама могут заключаться в использовании фильтрации технологий на основе распознавания шаблонов контента, исключений доменов, исключений адресов и/или фактического приведения в нерабочее состояние пролиферированных спамовых серверов посредством правоприменения. В режиме, в котором RBK-шифрование может представлять собой способ обмена данными по умолчанию, спам может обнаруживаться более детерминированным способом.

Одно из основных препятствий в способе автоматизации таких процессов, как RBK, может представлять собой существенную нехватку удобных для пользователя, доступных для пользователя и/или управляемых пользователем персональных приложений на основе инфраструктуры открытых ключей (PKI). NUTbook наряду с использованием Nut-контейнеров может пытаться заполнять PKI-пробел. Он может предоставлять гибкие, защищенные и/или управляемые пользователем способы для того, чтобы хранить, манипулировать и осуществлять доступ к этой информации прозрачным способом.

Фиг. 114 показывает блок-схему последовательности операций способа для того, чтобы уменьшать спам между Alice и Bob, которые теперь могут устанавливать RBK-канал связи, и он может представлять собой их способ связи по умолчанию, и они могут использовать известные открытые адреса электронной почты. Если электронная почта шифруется через RBK между Alice и Bob, то она, вероятно, может пред-

ставлять собой допустимую электронную почту от Alice к Bob или наоборот 11414. Если любой пользователь получает от другого электронное письмо, не зашифрованное с помощью RBK, то оно может с наибольшей вероятностью представлять собой спам и может отфильтровываться и может сохраняться в корзине 11412 для спама.

Фиг. 115 показывает блок-схему последовательности операций способа для того, чтобы уменьшать спам между Alice и Bob, которые теперь могут устанавливать RBK-канал связи, и он может представлять собой их способ связи по умолчанию, и они могут использовать неразглашенные личные адреса электронной почты (анонимные адреса электронной почты). Если электронная почта шифруется через RBK между Alice и Bob, то она, вероятно, может представлять собой допустимую электронную почту от Alice к Bob или наоборот 11514. Если любой пользователь получает от другого электронное письмо, не зашифрованное с помощью RBK, то оно может с наибольшей вероятностью представлять собой спам и может отфильтровываться и может сохраняться в корзине 11512 для спама. Этот пример может задаваться при условии, что набор личных адресов электронной почты может использоваться между Alice и Bob только для того, чтобы отправлять друг другу RBK-зашифрованные сообщения, за счет этого также расширяя концепт RBK-канала до уровня адреса электронной почты. Этот тип ориентированных на канал связи адресов электронной почты может задаваться в качестве анонимных адресов электронной почты.

Канал связи между Alice и Bob, которые могут согласованно использовать RBK через анонимные адреса электронной почты, может демонстрировать определенные характеристики, которые могут анализироваться, чтобы определять работоспособность самой взаимосвязи. Возможно, уже удалены некоторые, или все незашифрованные спамовые сообщения из канала по умолчанию, как описывается на фиг. 115. Теперь можно анализировать контекст надлежащих RBK-зашифрованных сообщений. Таблица на фиг. 116 перечисляет матрицу состояний на основе детерминированного контекста работоспособности канала связи Alice-Bob. Она может требовать качественной оценки контента посредством Alice, чтобы выяснять то, что может происходить с их взаимосвязью. Это показывает одностороннюю матрицу действий посредством Alice, которая может быть основана на поведении Bob, что подтверждается посредством его сообщений для Alice.

Последний симптом, перечисленный на фиг. 116, может приводить к интересному сценарию, когда роль Bob может заменяться посредством веб-производителя: т.е. Alice может устанавливать анонимный RBK-канал связи с производителем. Таблица на фиг. 117 показывает матрицу состояний на основе детерминированного контекста работоспособности канала связи Alice-производитель. Теперь, Alice может иметь способность отслеживать то, может или нет этот производитель продавать ее информацию спамерам, через канально-идентифицируемые аспекты анонимных адресов электронной почты и RBK-наборов. Это позволяет предоставлять уровень прозрачности относительно внутренних операций маркетингового отдела производителя с чистым журналом аудита. Этот тип учитываемости производителей может быть беспрецедентным таким систематически подробным способом посредством среднестатистического пользователя. Последствие для нарушения доверия Alice посредством производителя может быть страшным, поскольку производитель может потерять средство контакта с ней навсегда. Фактически, надлежащее и согласованное использование анонимных адресов электронной почты и/или RBK может предоставлять возможность цифрового эквивалента ухода Alice из магазина без возвращения; это может служить в качестве средства устрашения для производителей, чтобы не злоупотреблять персональной информацией своих клиентов.

Фиг. 118 показывает матрицу состояний на основе детерминированного контекста работоспособности канала связи Alice-производитель с точки зрения производителя. Характеристики канала могут предоставлять производителю идентичный тип одностороннего действия, которое он может предпринимать, чтобы защищать свой бизнес и возможно защищать своих клиентов. Использование этой технологии посредством производителя может, возможно, улучшать его репутацию в отношении конфиденциальности и безопасности данных. Оно также позволяет неявно утверждать, что производитель не может участвовать в массовой неразборчивой перепродаже клиентских данных.

Фиг. 119 показывает графическое представление того, как использование RBK может помогать изолировать компрометацию конфиденциальных данных в системе пользователя. Bob 11900 может иметь RBK-каналы с Alice 11910 и Dave 11920. Bob может щелкать на веб-узле с программой типа "троянский конь" и может заражаться регистратором нажатий клавиш или эквивалентной вредоносной программой, и после этого хакеры могут иметь возможность проникать в его защищенное хранилище данных для RBK, к примеру, в его NUTbook. Как результат, RBK-наборы Bob с некоторыми или всеми его контактами могут быть скомпрометированы 11900. Bob может контактировать с некоторыми или всеми своими друзьями, и он может уведомлять их относительно этого нарушения, либо некоторые его друзья могут уже логически выводить, что что-то не так Bob или его системой, из спамовых сообщений, которые могут отправляться им с использованием их закрытых каналов с Bob. Если посмотреть на NUTbook 11910 Alice, в котором она может сохранять некоторые или все свои RBK-наборы, она может помечать свой RBK-набор с Bob 11912 как скомпрометированный и может формировать новый RBK-набор каждый раз, когда Bob собирается с мыслями, чтобы удалять вирусы в своей системе. Это может наносить определенный ущерб для Alice, и он не распространяется на другие RBK-взаимосвязи, которые она может уста-

навливать. Это может быть полностью истинным, если она также согласованно использует анонимные адреса электронной почты со своими контактами. Alice может принимать спам от хакеров, но спам может игнорироваться автоматически, когда она помечает канал как скомпрометированный или удаленный. Когда Bob может быть готов, Alice может формировать новый набор RBK и новый анонимный почтовый канал, и они могут продолжать свой цифровой диалог в закрытом порядке. Процесс для Dave может быть идентичным для его RBK-хранилища 11920.

Анонимные взаимосвязи

Цифровые топологии и соглашения по взаимосвязям, которые могут возникать и цементироваться в Интернете за последние несколько десятилетий, могут быть неестественными и нереалистичными. Анонимность может представлять собой мощный конструкт взаимосвязей и может представлять собой уровень взаимосвязи, которой можно пользоваться на ежедневной основе с большинством случайных взаимодействий, таких как, но не только, хождение в аптеку для того, чтобы покупать персональные продукты, хождение в ресторан для того, чтобы покупать еду, вызов лицензионного такси подъемом руки для поездки и/или участие в митинге протеста. Вопреки этой физической реальности, почти каждый производитель в Интернете может хотеть знать точно, кем может быть Alice, в том числе и часть или всю персональную информацию, которую он может получать от нее. Множество производителей непосредственно могут оставаться относительно анонимными в силу непубликации прямых телефонных номеров и могут обслуживать клиентов через почтовые сообщения, системы проведения транзакций и/или удаленно привлеченных представителей по обслуживанию клиентов в удаленных центрах обработки вызовов. Наиболее преобладающее использование анонимности может представлять собой использование теми, кто может хотеть скрываться, к примеру, хакерами. В данный момент, может быть предусмотрено множество веб-узлов формирования поддельных личностей для людей, которые могут хотеть оставаться анонимными в Интернете, но они, возможно, должны отслеживать анонимность очень трудоемким способом и, возможно, должны принимать ответственные решения для того, чтобы быть намеренно двуличными. Использование RBK и анонимных адресов электронной почты может привнести некоторый паритет в этот дисбаланс анонимности в Интернете для среднестатистического пользователя и может уполномочить его иметь более значимую двунаправленную взаимосвязь с производителями и друг с другом без необходимости прибегать к поддельным персонам и случайной двойственности.

Фиг. 120 показывает упрощенный схематический вид предварительно скомплектованных персональных Nut-контейнеров данных. Nut-контейнер может сохранять детализированную персональную информацию относительно пользователя 12000. Он может автоматизировать предварительную комплектацию различных поднаборов этой персональной информации в различных целях. 12010 может представлять собой простой персональный Nut-контейнер данных, который может содержать только имя и адрес электронной почты. Анонимный персональный Nut-контейнер 12020 данных может показывать только альтернативный адрес электронной почты. Персональный Nut-контейнер 12030 данных покупок может включать в себя информационные поля, типично необходимые для веб-узлов покупок, чтобы покупать товары. Формирование этих поднаборов данных из главной информации 12000 может осуществляться через простые правила и фильтры и может формироваться по запросу во время процесса регистрации в Интернете. Независимо от того, может либо нет производитель или услуга подтверждать Nut-контейнеры данных, информация может задаваться доступной для вставки в корректные поля при необходимости посредством другого средства. Если пользователь использует преимущество анонимной почтовой услуги (прямая ссылка), Nut-контейнеры данных, такие как 12020, могут предлагать динамически созданные анонимные адреса электронной почты для конкретной устанавливающейся взаимосвязи.

Фиг. 121 иллюстрирует в виде диаграммы последовательность событий в процессе автоматизированной регистрации, который может использовать Nut-контейнеры. Производитель в Интернете может использовать и подтверждать персональные Nut-контейнеры данных и может предоставлять возможность установления RBK-каналов со своими клиентами автоматизированным способом. Пользователь может посещать вебсайт производителя и может хотеть регистрироваться 12100. Пользователь может начинать процесс посредством инструктирования NUTbook автоматически регистрироваться на веб-узле производителя, и может вводить URL-адрес регистрационного веб-узла. NUTbook может выполнять запрос к производителю, чтобы осуществлять выборку информации в отношении того, что производитель, возможно, должен регистрировать его 12102. NUTbook может составлять поднабор его персональной информации, которую может запрашивать производитель, и может показывать ему предварительную версию. Он может решать, что информация, запрашиваемая для регистрации, может быть приемлемой, и что NUTbook может собирать релевантную информацию и может продолжать процесс 12104. NUTbook может извлекать и может создавать предварительно скомплектованный Nut-контейнер, содержащий предварительно просмотренную информацию, и может отправлять его на веб-узел производителя. Производитель может подтверждать запрос на регистрацию и может отправлять запрос в адрес электронной почты пользователя, указываемый в предварительно скомплектованном Nut-контейнере 12106. Пользователь может принимать запрос производителя на свою электронную почту, запрашивающий его предоставлять свидетельство того, что он не представляет собой веб-робота, который может участвовать в несерьезной регистрации, посредством запрашивания его перехода по конкретному URL-адресу, чтобы

вводить капчу 12108 или другую форму возможной верификации. После того, как капча успешно вводится, производитель может удовлетворяться тем, что запрос может исходить от человека, и может продолжать устанавливать автоматически сформированные учетные данные для входа в учетную запись, ключ входа в учетную запись и/или RBK-наборы с пользователем. NUTbook пользователя может автоматически создавать пропускную карту для производителя, ее применимую веб-информацию, учетные данные для входа в учетную запись, ключ входа в учетную запись и/или RBK-набор 12112. Процесс регистрации может осуществляться во взаимодействии с пользователем в нескольких точках в процессе: инициирование, анализ/редактирование персонального комплекта данных и/или верификация на предмет человека. Трудности выбора имени для входа в систему, пароля, ввода в персональной информации и/или создания записи в хранителе паролей для производителя могут быть необязательными и могут быть автоматизированными. Когда его NUTbook активируется, он может иметь мгновенный доступ к производителю в полностью аутентифицированном режиме прозрачно, поскольку NUTbook может автоматически обеспечивать его вход в учетную запись в случае необходимости согласно упорядочению. Следует отметить, что этот процесс может осуществляться с любым производителем, приспосабливающим эту технологию для возможной выгода как пользователя, так и производителя. Меньше трудностей для пользователя и производителя позволяют получать более точную информацию от пользователя для базы данных и возможно вероятность большего числа транзакций между ними.

Фиг. 122 иллюстрирует в виде диаграммы последовательность событий в процессе автоматизированной регистрации с использованием Nut-контейнеров и анонимного адреса электронной почты. Производитель в Интернете может использовать и может подтверждать Nut-контейнеры и может предоставлять возможность установления RBK-каналов со своими клиентами автоматизированным способом с использованием анонимных адресов электронной почты. Пользователь может посетить веб-сайт производителя и может хотеть регистрироваться 12200. Пользователь может начинать процесс посредством инструктирования своей NUTbook автоматически регистрироваться на веб-узле производителя и может вводить URL-адрес регистрационного веб-узла. NUTbook может выполнять запрос к производителю, чтобы осуществлять выборку информации в отношении того, что производитель, возможно, должен его регистрировать 12202. Производитель может подтверждать анонимные регистрации, так что NUTbook может контактировать с NUTmail-услугой и может запрашивать пару анонимных адресов электронной почты под своей учетной записью. NUTbook может составлять и может показывать предварительный просмотр данных, которые должны отправляться в регистрацию производителя, которые могут включать в себя новые созданные анонимные адреса электронной почты. Он может решать, что информация, требуемая для регистрации, может быть приемлемой, и NUTbook может продолжать процесс 12204. NUTbook может создавать предварительно скомплектованный Nut-контейнер, содержащий предварительно просмотренную информацию, и может отправлять его на веб-узел производителя. Производитель может подтверждать запрос на регистрацию и может отправлять запрос на новый анонимный адрес электронной почты пользователя, указываемый в предварительно скомплектованном Nut-контейнере 12206. Пользователь может принимать запрос производителя на свой анонимный адрес электронной почты, запрашивающий его предоставлять свидетельство того, что он не представляет собой веб-робота, который может участвовать в несерьезной регистрации, посредством запрашивания его перехода по конкретному URL-адресу, чтобы вводить капчу 12208 или другую форму возможной верификации. После того, как капча успешно вводится, производитель может удовлетворяться тем, что запрос может исходить от человека, и может продолжать устанавливать автоматически сформированные учетные данные для входа в учетную запись, ключ входа в учетную запись и/или RBK-наборы с пользователем. NUTbook пользователя может автоматически создавать пропускную карту для производителя, его применимую веб-информацию, учетные данные для входа в учетную запись, ключа входа в учетную запись, анонимные адреса электронной почты и/или RBK-набор 12212. Процесс регистрации может осуществляться во взаимодействии с пользователем в нескольких точках в процессе: инициирование, анализ/редактирование персонального комплекта данных и/или верификация на предмет человека. Трудности выбора имени для входа в систему, пароля, ввода в персональной информации, создания адресов электронной почты и/или создания новой записи в хранителе паролей для производителя могут быть необязательными и могут быть автоматизированными. Когда его NUTbook активируется, он может иметь прозрачный доступ к производителю в полностью аутентифицированном режиме, поскольку NUTbook может автоматически обеспечивать его вход в учетную запись в случае необходимости согласно упорядочению. Этот процесс может не требовать персональной пользовательской информации и адресов электронной почты, которые могут создаваться, в частности, для этой взаимосвязи, что подразумевает то, что только релевантные почтовые сообщения между пользователем и производителем могут поступать на эти анонимные адреса почтовых сообщений. Поскольку ниже поясняются различные услуги на основе Nut-контейнеров, некоторые или все из них предлагают анонимные регистрации.

Каналы связи, которые могут устанавливаться с использованием RBK и анонимных адресов электронной почты, могут минимизировать спам детерминированным способом вследствие своего режима шифрования по умолчанию всего через RBK. Кроме того, они могут предоставлять двунаправленное управление каналом для сторон, которые могут участвовать, так что может быть взаимоуважение к взаи-

мосвязи и ее подразумеваемым ограничениям. Отклонения от этих подразумеваемых границ взаимосвязей позволяют точно определять события изменения взаимосвязей и могут обусловливать одностороннюю реакцию в диапазоне от запросов до полного разрыва взаимосвязи детерминированным способом. Для третьих сторон, пытающихся расшатывать данные Bob или Alice, помимо извлечения корректной пары анонимных адресов электронной почты, третья сторона, возможно, также должна взламывать зашифрованные сообщения и документы.

Веб-узлы, которые могут подтверждать и могут обрабатывать автоматизированные регистрации, могут добавлять дополнительные услуги, такие как, но не только, фильтрация по возрасту. Родители могут депонировать предварительно скомплектованный Nut-контейнер на NUTserver устройства своего ребенка, чтобы указывать некоторые общие признаки идентификационных данных, таких как, но не только, пол, возраст и/или общее местоположение. Этот предварительно скомплектованный Nut-контейнер может автоматически использоваться для того, чтобы регистрировать ребенка в любом благоприятном для детей или предварительно одобренном родителями веб-узле, который может подтверждать Nut-контейнеры. Производитель может утверждать или отклонять попытки доступа на основе этой информации и услуг, которые они могут предоставлять, таких как, но не только, веб-узлы по продаже алкоголя, веб-узлы по продаже сигарет, веб-узлы киноанонсов, веб-узлы только для взрослых и/или веб-узлы по продаже огнестрельного оружия. Кроме того, Nut-контейнер для регистрации Интернет-активности может быть сконфигурирован на NUTserver устройства ребенка, чтобы отслеживать его активность и цифровое местонахождение. Ограничения на Интернет-использование также могут администрироваться родителем посредством использования таких Nut-контейнеров на некоторых или всех устройствах в доме таким образом, что коммутация устройств может быть несущественной в отношении совокупного использования Интернета ребенка в день. Блокирование или разрешение на доступ к определенным веб-узлам может осуществляться посредством использования таких Nut-контейнеров идентификации детей на самом устройстве и/или в сочетании с конкретными конфигурационными настройками на Wi-Fi-маршрутизаторе на основе NUTS (прямая ссылка).

Базовые NUTS-приложения

Таблица на фиг. 123 перечисляет приложения, которые могут содержать набор базовых NUTS-приложений. Эти приложения могут постоянно размещаться в большинстве систем, которые могут использовать NUTS-технологии, и они могут обрабатывать Nut-файлы, как показано на этой упрощенной схеме функционального вычислительного устройства на фиг. 124. Как отмечено выше, некоторые или все эти приложения могут уже упоминаться посредством материала, поясненного выше в этом раскрытии сущности. Эти приложения могут не детализироваться заранее в этом раскрытии сущности вследствие своих зависимостей от некоторых или всех базовых основополагающих функций и характеристик NUTS, таких как, но не только, узлы замка, графы замков, Nut-части, Nut-предыстория, Nut-журнал регистрации, MIO, MIOR, Nut-идентификаторы, RBK, непрозрачность градиента и/или анонимные взаимосвязи. Некоторые или все эти базовые приложения могут предпочитать использовать Nut-контейнер в качестве базовой единицы хранения, которая может быть осуществлена посредством обычного файла, но не только. Это может подразумевать, что некоторые или все данные, с которыми эти системы соприкасаются, сохраняют и/или манипулируют, может поступать с высокой степенью безопасности и управления доступом по умолчанию. Принципы философии проектирования, которые могут использоваться в проектировании на основе узлов замка, которые могут помогать читателю в понимании этих базовых приложений более полно, могут представлять собой концепты итерации, интеграции, независимости и/или идентифицируемости.

Базовое NUTS-приложение: NUTserver NUTserver может быть проиллюстрирован схематично на упрощенной схеме пользовательского устройства на фиг. 125. Может быть предусмотрено несколько функций ключей, которые NUTserver может выполнять в фоновом режиме, чтобы организовывать и поддерживать NUTS-совместимое окружение. NUTserver 12510 может выполняться в пространстве приложений пользовательского вычислительного устройства 12500. Устройство может иметь некоторое хранилище 12520, в котором могут сохраняться Nut-файлы 12522. NUTserver может отвечать за предоставление API и каналов связи, открытых с различными приложениями, содержащими NUTbook 12512, NUTbrowser 12514 и/или другие приложения 12516, включающие в себя ОС устройства. NUTserver также может отвечать за поддержание внешних соединений с другими устройствами, которые могут принадлежать пользователю, который может запускать NUTserver 12530 и возможно может взаимодействовать с NUTcloud 12540. NUTserver может не быть заменой для файловой системы пользовательского устройства 12500, а вместо этого может работать через локальную операционную систему и файловую систему, чтобы осуществлять доступ и обрабатывать любые Nut-файлы.

Фиг. 126 показывает упрощенную схему основных внутренних частей NUTserver и его функциональностей. Пользовательское устройство может иметь операционную систему 12600, управляющую аппаратными средствами и программным обеспечением. Устройство может иметь внешнюю связь, обслуживаемую посредством сетевых интерфейсов 12602 и их ассоциированных драйверах, работающих на ОС 12600. Устройство также может иметь файловую систему 12604, которая может присоединяться и может управляться посредством ОС 12600. В файловой системе могут быть сохраняться хранилища дан-

ных для MIOR 12610, и пользовательские данные могут содержаться в Nut-контейнерах 12606 и 12608. ОС 12600 также может выступать в качестве прикладного окружения, в котором могут выполняться множество приложений, содержащих приложения, проиллюстрированные на схеме: NUTserver 12620, NUTbook 12642, NUTbrowser 12644, MIOR-сервер 12646 и другие приложения 12648. NUTserver 12640 может выполняться на другом устройстве, но прикладной интерфейс 12636 также может обрабатывать эту связь.

В NUTserver 12620, может быть предусмотрен собой модуль 12622, который может выполнять аутентификации в NUTserver и может поддерживать кэш ключей. Когда NUTserver запускается, он может не иметь полномочий взаимодействовать с какими-либо защищенными уровнями в любых Nut-контейнерах. Пользователь и/или аппаратные средства могут предоставлять необходимую аутентификацию, которая может обеспечивать возможность модулю 12622 NUTserver-аутентификации получать доступ к определенным наборам ключей. Это может заключаться просто в наличии защищенного фразовым паролем Nut-контейнера, хранящего наборы ключей, и в запрашивании пользователя на предмет того, чтобы предоставлять фразовый пароль, открытии Nut-контейнера и кэшировании в защищенное/незащищенное запоминающее устройство наборов ключей в рабочих данных; или они могут представлять собой защищенные аппаратно-предоставляемые ключи, содержащиеся на множестве вычислительных устройств; или они могут представлять аппаратный маркер, такой как, но не только, USB-ключ, который может предоставлять пользователь. Набор ключей может содержать минимум ключ NUTserver-аутентификации и/или ключ для каждого базового NUTS-приложения, которое может устанавливаться на локальном устройстве. Может быть кэш 12624, который может поддерживаться посредством NUTserver в организационных целях и для эффективности. Часть кэша может представлять собой индекс 12626 Nut-идентификаторов. Этот индекс может содержать некоторые или все Nut-идентификаторы, которые пользователь может хотеть отслеживать локально и удаленно. Поиск Nut-идентификатора в индексе может указывать то, где может содержаться Nut-идентификатор. Другая часть кэша 12624 может низводиться до хранения Nut-кэша 12628 в запоминающем устройстве для Nut-контейнеров с частым осуществлением доступа.

NUTserver может отвечать за синхронизацию контента двух или более Nut-контейнеров с идентичными Nut-идентификаторами 12630. После того, как NUTserver может быть надлежащим образом аутентифицирован, и он может иметь достаточные ключи, чтобы осуществлять доступ к некоторым или всем Nut-контейнерам, принадлежащим пользователю, далее он может открывать различные Nut-контейнеры, чтобы анализировать их контент и управлять ими. Каждый Nut-контейнер может хранить номер версии и временную метку последнего обновления или модификации. Если обновление возникает в Nut-контейнере, и NUTserver может уведомляться в отношении него, или NUTserver может замечать это, то он может помечать обновление и может искать индекс 12626, чтобы видеть некоторые или все местоположения, в которых копия этого обновленного Nut-контейнера может существовать локально или удаленно. После этого он может систематически начинать распространять и синхронизировать 12630 изменения затрагиваемых Nut-контейнеров.

Этот процесс может быть достаточно простым вследствие метаданных, встроенных внутри каждого Nut-контейнера, таких как, но не только, Nut-идентификатор, номер версии, внутренние ц-подписи, предыстория и/или журнал регистрации. Новейшая версия может просто перезаписывать существующую версию, если могут удовлетворяться различные критерии модификации. Может быть необязательным, что NUTserver должен иметь возможность взаимодействовать с Nut-контейнером частично или полностью, поскольку он может зависеть от просматриваемых метаданных, которые могут разрешаться посредством непрозрачности градиента Nut-контейнера в отношении того, может или нет осуществляться синхронизирующее обновление. Достаточные метаданные открытого текста могут обеспечивать возможность синхронизации некоторых Nut-контейнеров посредством NUTserver без ключей для рассматриваемых Nut-контейнеров. В случаях, если может возникать вероятность разветвления или ветвления версий, пользователь может вовлекаться в решение касательно того, какую версию задавать в качестве текущей. Функция 12630 репликации может обеспечивать возможность равноправным NUTserver распространять эти типы изменений на управляемые пользователем устройства автоматически. Функциональности, предоставленные посредством 12630, могут составлять персональный NUTcloud для пользователя, когда он может устанавливать и соединять несколько NUTserver на своих устройствах. Он может пользоваться синхронизированными и/или реплицированными Nut-контейнерами на любом из своих устройств автоматизированным способом. Когда возникают более сложные проблемы версий, или определенная статистическая версия Nut-контейнера может запрашиваться, модуль 12632 управления исправлениями может обрабатывать эти запросы. Он может использовать конкретные дельта-способы поиска версии, используемые посредством Nut-контейнера, и может выполнять большую детализацию управления версиями, чтобы формировать требуемую версию Nut-контейнера. Эти конкретные для Nut-контейнера дельта-способы поиска версии и способы чтения/записи контента Nut-контейнеров могут существовать или не могут существовать в локальном MIOR, так что может быть предусмотрен MIOR-интерфейс 12634, чтобы предоставлять эти функции, когда они могут требоваться.

Nut-контейнер доступа может задаваться как защищенный Nut-контейнер, который может содер-

жать аутентификационные учетные данные для других систем или контейнеров, такие как, но не только, входы в учетную запись веб-узла, входы в учетную запись базы данных, корпоративные системы, персональные устройства, программные системы, другие Nut-контейнеры, NUTserver, почтовые системы, чат-системы и/или любая цифровая система, требующая секретного парольного ключа доступа и/или идентификатора для входа в систему. NUTserver может представлять прикладной интерфейс 12636 для других приложений, чтобы осуществлять доступ к его функциям. NUTserver может идентифицироваться посредством своего типа приложения и подробностей установки, дополнительно ему также может назначаться Nut-идентификатор. Конфигурационный NUTS-файл для пользовательского устройства может указывать на каталог или область конфигурации в файловой системе 12604, в которой он может находить Nut-контейнер доступа, хранящий информацию для каждого приложения, который он, возможно, должен знать, такую как, но не только, удаленные и/или локальные NUTserver. Например, локальный каталог NUTserver 12620 конфигурации может хранить Nut-контейнер доступа, содержащий Nut-идентификатор, тип и/или ключи доступа для удаленного NUTserver 12640. Успешно открытие такого Nut-контейнера доступа может предоставлять локальному NUTserver 12620 достаточную информацию для того, чтобы пытаться контактировать с удаленным NUTserver 12640 и аутентифицироваться на нем таким образом, что он может открывать доверенный канал связи и отправлять друг другу Nut-контейнеры. Аналогично, могут быть предусмотрены конфигурационные Nut-контейнеры для различных приложений, с которыми может взаимодействовать NUTserver. Поскольку Nut-контейнеры доступа представляют собой Nut-контейнеры, они могут сохраняться синхронизированными, реплицированными и/или распространяемыми между равноправными NUTserver.

Из этого пояснения того, как NUTserver может функционировать, итеративный проектный подход для внутренних элементов Nut-контейнера может расширяться на то, как приложения и данные, ассоциированные с возможностью конфигурировать и аутентифицировать их, могут сохраняться и быть доступными. Конфиденциальные данные могут сохраняться в Nut-контейнере в максимально возможной степени. Последствия такого простого утверждения становятся далеко идущими при рассмотрении встроенных функций и признаков Nut-контейнера и функций, предоставленных посредством NUTserver. Неаутентифицированный NUTserver может предоставлять достаточную функциональность для того, чтобы реплицировать, распространять и/или синхронизировать Nut-контейнеры, к которым он может не иметь внутреннего доступа. Это может быть обусловлено свойством непрозрачности градиента Nut-контейнера: множество Nut-частей, составляющих нераскрываемые метаданные, могут сохраняться в качестве открытого текста и могут предоставлять достаточную информацию для множества нормальных действий по обслуживанию, которые должны выполняться для Nut-контейнера посредством NUTserver. Вследствие признаков обеспечения безопасности, которые могут быть встроены в Nut-контейнер, безопасность каналов связи для транспортировки Nut-контейнеров между приложениями через WAN или сеть intranet может иметь меньшую значимость.

Этот способ использования Nut-контейнеров доступа может разрешать многочисленные проблемы, ассоциированные с проектированием программного обеспечения, программированием и/или использованием. Например, главная проблема разработчиков программного обеспечения может возникать, когда они жестко кодируют входы в учетную запись и пароли в своем коде в процессе разработки своего кода, чтобы ускорять запись в тестовую систему, к примеру, в тестовую базу данных или в сервер тестовых приложений. Переход в QA- и рабочие режимы тестирования и разработки может осуществляться посредством добавления дополнительных процедур аутентификации в код непосредственно перед этой стадией, которая может минимально тестироваться. При использовании Nut-контейнеров доступа, может быть возможным интегрировать их в программу разработки на ранних стадиях, и процесс, возможно, никогда не должен изменяться, а может изменяться только Nut-контейнер доступа. Диспетчер может назначать и создавать соответствующие Nut-контейнеры доступа для разработчика, инженера по контролю качества и/или пользователя формирования. Эти Nut-контейнеры доступа могут эффективно интегрироваться в их соответствующие NUTbook-совокупности и могут обеспечивать им возможность соединяться с их ресурсами приложений вообще без отдельного входа в систему. Диспетчер может фактически поддерживать владение Nut-контейнеров доступа и изменять его по мере необходимости, и NUTserver могут в конечном счете реплицировать и/или синхронизировать его таким образом, что конечные пользователи, возможно, никогда не должны беспокоиться касательно этого, в силу чего руководитель проекта может удаленно и защищенно управлять взаимосвязями между пользователями и их приложениями. Эффективное использование Nut-контейнеров доступа может обеспечивать возможность пользователю конфигурировать свои системы для единой точки входа (SSO): SSO на их локальном NUTserver, а все остальное может автоматически аутентифицироваться при необходимости. Иерархические пароли (прямая ссылка) могут предоставлять возможность дополнительной защиты для определенных поднаборов доступа и информации.

Фиг. 127 является альтернативным вариантом осуществления NUTserver, в котором Nut-кэш 12628 может заменяться посредством функциональностей NoSQL-базы 12728 данных. NoSQL-базы данных могут рассматриваться некоторыми в качестве поднабора объектно-ориентированных баз данных, и многие из них могут быть очень эффективными при обработке Nut-образных контейнеров, которые могут

представлять собой нетабличные структуры. Некоторые NoSQL-базы данных, такие как CouchBase, могут предлагать встроенную репликацию и другие признаки, которые могут использоваться посредством NUTserver, чтобы выполнять некоторые из его обязанностей.

Базовое NUTS-приложение: MIOR-сервер

Модульный репозиторий ввода-вывода или MIOR может представлять собой серверную услугу, как проиллюстрировано на фиг. 128. Он может представлять собой типичный вариант осуществления MIO-систем и способов. Вычислительное устройство 12810 может иметь локальный MIOR-сервер, выполняющийся на устройстве с собственным локальным MIOR-кэшем 12812. Если запрос не может удовлетворяться посредством локального MIOR-сервера, он может обратиться к известным MIOR-серверам 12820 на основе Интернета или их зеркалам 12830. В их соответствующих кэшах 12822 и 12832 можно выполнять поиск на предмет адаптированных MIO-модулей в запросе. Если содержатся, он может отправлять их обратно в инициирующий MIOR-сервер на вычислительном устройстве пользователя. Если запрашиваемые модули не содержатся на первом MIOR-сервере 12820 в Интернете, MIOR-сервер 12820 может обратиться к другим MIOR-серверам в Интернете, чтобы искать их. Исходный запрос может иметь тайм-аут или каскадный предел на число каскадируемых запросов, которые он может выполнять всего. В некоторых вариантах осуществления, запросы могут осуществляться асинхронно, а не в режиме блокирования в подходящих случаях.

Более тесная проверка этого процесса может быть проиллюстрирована на фиг. 129. Приложение 12918 может выполняться на локальном устройстве 12910, которое, возможно, должно читать Nut-файл 12908 в свое запоминающее устройство. Nut-контейнер 12908 может указывать то, что ему может требоваться определенный набор модулей чтения и записи для его рабочих данных из MIOR-сервера 12914. Приложение может контактировать со своим локальным MIOR-сервером 12914 и может запрашивать модули чтения и записи на предмет этого Nut-контейнера. MIOR-сервер 12914 может проверять в своем локальном MIOR-кэше 12916 то, может он или нет иметь эти модули. Если содержатся, он может отвечать обратно в приложение с модулями или информацией местоположения модулей в локальной системе или сети. Если не содержатся, MIOR-сервер 12914 может охватывать WAN 12900 или другую сеть MIOR-серверов, чтобы запрашивать их из более крупного MIO-репозитория, такого как 12920. MIOR-сервер 12920 может представлять собой выделенный сервер, оптимизированный с возможностью обслуживать запросы из Интернета для различных модулей. После того, как MIOR-сервер 12922 принимает запрос из MIOR-сервера 12914, он может проверять свой локальный MIOR-кэш 12924 на предмет этих модулей. Если содержатся, он может отвечать обратно на MIOR-сервер 12914 с модулями в запросе. Если не содержатся, он может контактировать с другими MIOR-серверами в своей группе равноправных узлов при поиске этих модулей. Между тем, он может отправлять сообщение "Не могу найти, но продолжаю поиск" обратно в MIOR-сервер 12914. Когда удаленный запрос возвращается с запрашиваемыми модулями, локальный MIOR-сервер 12914 может аутентифицировать их до сохранения их в свой локальный MIOR-кэш 12916. Как всегда, когда настает время для приложения 12918, чтобы создавать экземпляр и использовать модуль, он также может аутентифицировать контент с использованием нормальных внутренних NUTS-механизмов.

Фиг. 130 показывает блок-схему последовательности операций способа для выборки MIO-модулей из MIOR-сервера.

Аутентификация между удаленным MIOR-сервером и локальным MIOR-сервером может устанавливаться через сеансовые ключи или анонимные учетные записи при желании. Более высокие уровни обслуживания могут включать в себя доступ к монопольным модулям с настраиваемыми Nut-контейнерами с ключом, к примеру, корпорация может хотеть использовать широкое распределение MIOR-сети для своих сотрудников с использованием индивидуально разработанного программного обеспечения, но сотрудники могут открывать и аутентифицировать настраиваемые модули только в том случае, если они имеют есть ключ доступа возможно в Nut-контейнере доступа из компании таким образом, внутренняя информация может защищаться согласованно на относительно открытой платформе предоставления услуг.

Типичный вариант осуществления внутренней организации MIOR-кэша показан на фиг. 131. Кэш 13100 может иметь набор индексов 13110, который может содержать ссылку на различные модули, которые могут снабжаться перекрестными ссылками и индексироваться. Структура MIOR не ограничена этим вариантом осуществления, но может содержать некоторые или все эти организационные структуры и технологии. Поскольку каждый модуль может сохраняться в Nut-контейнере, главный индекс 13112 Nut-идентификаторов может содержать некоторые или все Nut-идентификаторы модулей и их местоположений в кэше. Индексы 13114 модулей файлового ввода-вывода могут перечислять некоторые или все модули этого типа по описанию и Nut-идентификатору. Индекс 13118 модулей файловых приложений может перечислять некоторые или все модули этого типа по описанию и Nut-идентификатору. Индекс 13120 модулей отображения файлов может перечислять некоторые или все модули этого типа по описанию и Nut-идентификатору. Индекс 13116 модулей совокупностей может перечислять некоторые или все модули, принадлежащие совокупности, по описанию и Nut-идентификатору. Могут компоноваться другие индексы, чтобы предоставлять возможность эффективного выполнения поиска кэша. Группы 13130-

13150 совокупностей (прямая ссылка) проиллюстрированы на схеме, чтобы визуально показывать то, как связанные модули могут группироваться. Способ группировки совокупностей может выполнять важную роль в операциях NUTbook.

Базовое NUTS-приложение: NUTbrowser/NUTshell

Фиг. 132 показывает схему NUTbrowser-приложения. NUTbrowser по существу может представлять собой графический пользовательский интерфейс (GUI), который может выполняться поверх функциональностей NUTshell-приложения с интерфейсом командной строки (CLI). Общеизвестные программы-оболочки могут представлять собой bash-оболочку, csh, cmd.exe, DOS-оболочку, в числе других. Общеизвестные программы файлового менеджера могут представлять собой Windows Explorer, Apple Finder и другие. Направленное на пользователя поведение этих двух программ может быть почти идентичным их общеизвестным дубликатам; тем не менее, отличие может заключаться в том, что NUTbrowser и NUTshell могут распознавать Nut-контейнеры и могут обрабатывать их более полно, чтобы использовать преимущество обширных метаданных, которые могут сохраняться в каждом Nut-файле. Каждый Nut-файл может идентифицироваться посредством двух способов: поверхностное расширение имени файла "*.nut" и/или более глубокое тестирование сообщениями контента в качестве Nut-контейнера. Большинство файловых систем могут приспосабливать способ на основе расширений имен файлов. Попытка Nut-чтения может использоваться при попытке подтверждать то, что *.nut-файл фактически может представлять собой Nut-контейнер, либо при введении новых файлов в локальную систему из недоверенного источника.

Большинство популярных операционных систем, таких как Mac OS, Windows и/или Linux, могут использовать несколько способов, чтобы идентифицировать тип файла, содержащие расширения имен файлов, магические числа, универсальные идентификаторы типов (UTI), атрибуты файловой системы и т.п. Расширения имен файлов могут представлять собой наиболее поверхностный способ, поскольку, когда имя файла может изменяться, связь между типом контента и распознаванием может разрываться. Магические числа и UTI могут представлять собой компактные, но ограниченные формы метаданных, встроенных в заголовок файла, и могут требовать доступа к индексу типов файлов для того, чтобы перекрестно ссылаться на то, какую форму может иметь контент. Этот индекс типов файлов может существовать в ОС, файловой системе или другой внешней системе. Атрибуты файловой системы могут представляться как атрибуты объекта файла, которые могут присоединяться к его экземпляру в механизме индексации файловой системы. Эта информация может быть эффективной только в пределах домена комбинации файловой системы/операционной системы, которая может записывать и распознавать ее. Nut-метаданные могут не только указывать тип рабочих данных, но и то, как они могут читаться, записываться, отображаться, и/или выполнять их. Они могут указывать некоторые или все версии различных модулей, которые могут требоваться для того, чтобы успешно обрабатывать контент. Фактически, они могут удалять некоторые или все зависимости во всех без исключения таблицах внешних ссылок для обработки контента, таких как, но не только, записи системного реестра Windows и/или списки свойств Mac OS. Это может обеспечивать возможность Nut-контейнеру самоописывать и предписывать необходимые компоненты, которые могут требоваться для того, чтобы осуществлять доступ к его контенту, и может обеспечивать возможность MIOR-серверу автоматически устанавливать любые компоненты, которые могут отсутствовать во время доступа.

NUTbrowser/NUTshell может читать метаданные любого выбранного Nut-контейнера и может обмениваться данными с различными другими базовыми приложениями NUT-контейнера, чтобы пытаться открывать, отображать и/или выполнять надлежащее приложение в контенте Nut-контейнера посредством осуществления доступа 13232 к MIOR-серверу 13250. Если пользователь надлежащим образом аутентифицирован в NUTserver 13240, NUTbrowser/NUTshell может иметь доступ 13234 к некоторым или всем необходимым Nut-контейнерам доступа для того, чтобы надлежащим образом открывать Nut-контейнеры еще больше. Фактически, NUTbrowser/NUTshell может действовать не отличающимся образом по сравнению с любым приложением, которое может надлежащим образом обрабатывать Nut-контейнер.

В зависимости от постоянного хранилища, которое может использоваться в локальной системе, NUTbrowser/NUTshell может обеспечивать возможность нескольким Nut-контейнерам с идентичным именем файла существовать в идентичной области хранения при условии, что Nut-идентификаторы могут отличаться. Некоторые системы хранения данных, такие как базы данных и объектные файловые системы, могут не быть чувствительными к именам файлов. Для большинства облачных систем хранения, способ идентификации на основе Nut-идентификаторов может подходить в большей степени, чем традиционные способы на основе имен путей.

Базовое NUTS-приложение: NUTbook

Схематический вид NUTbook показан на фиг. 133. К настоящему моменту, типичное приложение Nut-обработки может выглядеть знакомым с аналогичными компонентами; оно может формировать основу инфраструктуры Nut-обработки, более обобщенной на фиг. 134, и может функционировать аналогично тому, как может работать NUTbrowser-приложение на фиг. 132. NUTbook может иметь требуемые интерфейсы с NUTserver 13334 и MIOR-сервером 13332. Оно может обрабатывать MIOR-модули 13326-

13330 по мере необходимости, чтобы предоставлять функциональности, предоставленные посредством них, как указано посредством 13322 и 13324. Основная функция NUTbook может заключаться в том, чтобы поддерживать организованный набор 13336 кэшей, называемый карточным каталогом. NUTbook может представлять собой электронный карточный каталог, состоящий из совокупностей данных, как показано на фиг. 135. NUTbook может предлагать некоторые функциональности, содержащиеся в типичном персональном информационном менеджере. Почему NUTbook представляет собой карточный каталог? Вот список различных причин, по которым это может быть целесообразным:

Пользователи могут не иметь простого способа собирать, обрабатывать и организовывать произвольные наборы данных.

Обычно это может осуществляться неформально в электронных таблицах, текстовых файлах или простых базах данных.

Может не быть легкодоступной общей полезности в том, чтобы получать, организовывать и/или каталогизировать различные совокупности данных защищенным способом, при этом репозиторий может содержать файл данных в расчете на элемент в совокупности.

PKI-сертификаты, контактные карты, RBK-наборы, веб-входы в учетную запись, бейсбольная статистика, VPN-входы в учетную запись и учетные данные, предыстория пользования автомобилем, DVD-коллекции, коллекции марок, коллекции книг, детские медицинские карты и т.д. Они могут рассматриваться как различные совокупности данных или карт.

Nut-контейнер может защищенно сохранять каждый тип элемента защищенным способом, который может быть простым в использовании и транспортировка.

В силу этого можно сохранять некоторые или все ключи шифрования, которые также могут требоваться для того, чтобы инструктировать NUTS работать прозрачно в Nut-контейнеры.

Можно осуществлять доступ к этим совокупностям карт посредством индексации их Nut-идентификаторов и любых необязательных поисковых индексных метаданных в NUTbook-приложении.

NUTserver могут иметь сведения по определенным важным типам карт и могут приоритезировать свою обработку во множестве своих задач.

Nut-контейнер, который может существовать в окружении с несколькими NUT-серверами, может иметь репликацию, синхронизацию, регистрацию, полную предысторию, шифрование и/или управление доступом, по умолчанию скомплектованные в один файл в расчете на элемент для легкой транспортабельности.

NUTbook может содержать кэш 13520 ключей, который может иметь форму защищенного или незащищенного запоминающего устройства в зависимости от доступных аппаратных средств. Кэш ключей может сохранять часто используемые ключи доступа с присоединенными надлежащими атрибутами, такими как, но не только, число раз, когда он может использоваться до истечения срока, время истечения срока и/или события истечения срока действия. Его основной кэш 13550 каталогов может иметь главный индекс Nut-идентификаторов Nut-контейнеров, которые он может отслеживать. Кэш может состоять из различных совокупностей данных, таких как, но не только, PKI-сертификаты 13562, контактные карты 13564, NUTserver-карты 13566 доступа, карты 13568 управления документооборотом и/или любые другие заданные совокупности 13570. Эти совокупности могут сохраняться в запоминающем устройстве, в базе данных, в файловой системе или в другом механизме хранения данных в зависимости от конфигурации NUTbook и доступных аппаратных средств. База данных и хранилище файловой системы могут находиться удаленно при условии, что они могут быть локально доступными через сетевой интерфейс. Фиг. 136 может представлять собой пример схемы размещения того, как может организовываться кэш NUTbook-каталогов.

Данные, сохраненные в NUTbook, могут представлять собой агломерацию PIM, хранителя паролей, диспетчера PKI-сертификатов, связки ключей, адресной книги, приложения для создания заметок, книги рецепта, индекса CD-коллекции, индекса коллекции марок, индекса коллекции книг, медицинских карт и/или любых других наборов данных, которые могут выражаться как совокупность. Современный уровень техники для среднестатистического пользователя не может предлагать множество вариантов выбора для него, чтобы в цифровой форме организовывать различные фрагменты их жизней в функциональной цифровой форме. Приложения адресной книги могут быть многочисленными, но может недоставать прозрачной, легкой перекрестной совместимости. Большинство воспринимаемых пользователей не могут сохранять чувствительные пароли в своих адресных книгах и могут оценивать и использовать приложение хранителя паролей для этой конкретной цели. Даже только для этих двух простых приложений, адресной книги и хранителя паролей, если пользователь должен учитывать такие признаки, как совместимость операционных систем, синхронизация, облачные отпечатки, резервные копии, интеграция веб-браузера в числе других, матрица принятия решений может расширяться на несколько размерностей. Так же, может не быть гарантии хорошей интеграции между хранителем паролей и адресной книгой. Если пользователь хочет отслеживать медицинские карты своего члена семьи, записи автоматического обслуживания, расписания техобслуживания дома, входы в школьные учетные записи, связанные с занятиями детей, ветеринарные карты домашних животных, информацию цифровых устройств и/или другие совокупности данных, он, возможно, должен осуществлять это во всевозможных форматах с использованием

различных приложений для каждого типа данных. Общее использование электронных таблиц может заключаться в том, чтобы организовывать такие различные наборы данных и может выступать в качестве базы данных общего назначения для пользователя. NUTbook может обеспечивать возможность пользователю систематически сохранять некоторые или все типы информации в Nut-форме и может интегрировать использование данных в любое Nut-совместимое приложение. Данные, которые могут надлежащим образом формироваться и идентифицироваться, могут задаваться функциональными посредством приложений, которые могут использовать преимущество их заданной структуры. Некоторые или все признаки NUTS-окружения могут быть доступными для каждого Nut-контейнера в NUTbook, такие как, но не только, безопасность, синхронизация, репликация, резервное копирование и/или неустаревание.

Неустаревание и/или временная совместимость могут быть важной характеристикой использования MIOR. Посредством использования совокупностей в NUTbook наряду с MIOR, пользователь может получать несколько преимуществ: данные, которые он может формировать, могут быть его, они могут быть защищенными, и они могут иметь обоснованное ожидание иметь возможность осуществлять доступ к своим данным неограниченно (либо при условии, что NUTS может быть активным и поддерживаться). NUTbook также может выступать в качестве моста между миром пользователя базы данных и миром пользователя файла. Он может предоставлять выгоды базы данных в форме записей, сохраненных в формате файлов. MIO-модуль для функциональности чтения/записи для конкретной совокупности может представлять собой организованный набор спецификаций полей, связанных с захватом подробностей конкретной совокупности, которую может иметь в виду пользователь, но он может не быть ограничен этой моделью. В некоторых вариантах осуществления, модули чтения/записи могут представлять собой интерфейсы с различными базами данных и могут предоставлять функциональность увязки и преобразования полей для вызывающего приложения. В других вариантах осуществления, они могут представлять собой модули чтения/записи, которые расшифровывают собственные двоичные форматы рабочих данных с использованием лицензированных ключей из компании-разработчика программного обеспечения. Множество способов, которыми модули могут использоваться для того, чтобы осуществлять доступ к данным, могут быть очень разнообразными и могут иметь множество перестановок в зависимости от целей разработчика приложений. Базовая структура конкретной совокупности может индивидуально настраиваться пользователем с очень небольшими знаниями в области программирования, начиная с простых уже существующих шаблонов. Новые и полезные совокупности могут добавляться в локальный MIOR для его личного использования и совместно использоваться с другими через Nut-файлы. Они также могут отправляться в Интернет-MIOR-сервер для использования посредством любым после некоторого процесса утверждения.

Теперь, когда, возможно, раскрыты некоторые стимулы и проектные цели NUTbook, можно акцентировать внимание на том, как NUTbook может выступать в качестве PKI и в конечном счете может предлагать SSO-уровень обслуживания для среднестатистического пользователя. Фиг. 137 проясняется концепт иерархических паролей. В NUTS-диалекте, пароли могут быть эквивалентными фразовым паролям, поскольку Nut-контейнер может подтверждать обе формы, и вместо любого пароля, пользователь может использовать аппаратные маркеры, кодированные двоичные ключи или любой другой способ, который может предоставлять секретный ключ. Вычищаемая пролиферация паролей и их ассоциированных дифференциаторов, такая как, но не только, двухфакторные аутентификации, правила входа в учетную запись, настраиваемые правила установки пароля, настраиваемые вебстраницы и/или аппаратные маркеры, может быстро выходить из под контроля и может оставлять пользователя в психологическом состоянии, в котором он может прибегать к тому, чтобы чрезвычайно просто запоминать пароли для множества веб-узлов, в силу чего пользователь может противодействовать усилиям отдельных производителей обеспечивать большую защищенность своих систем для клиентов. Предпочтительное решение для NUTS может заключаться в том, чтобы использовать максимально небольшое количество паролей, чтобы обеспечивать возможность эффективного SSO-доступа, и иерархические пароли могут осуществлять этот подход. Может быть предусмотрен основной пароль 13710, который может обеспечивать возможность базовой аутентификации в NUTserver и NUTbook. Основной пароль может открывать Nut-контейнер, содержащий ключ, который может кэшироваться в кэше 13520 ключей, и может быть выполнен с возможностью автоудаляться после конца сеанса или предварительно определенного события. Этот основной ключ может быть достаточным для того, чтобы эффективно использовать большую часть NUTserver- и NUTbook-функций. Могут быть предусмотрены пароли второго уровня, такие как, но не только, покупки 13714, работа 13716, финансы 13718 и/или связь 13712. Эти пароли могут вводиться только после успешного ввода допустимого основного пароля, в силу чего они могут придерживаться иерархии паролей. Этот второй уровень может обеспечивать возможность пользователю выделять и изолировать разные уровни безопасности для различных групп данных. Каждый пароль на втором уровне может быть выполнен с возможностью иметь различную продолжительность действия в кэше 13520 ключей таким образом, что пользователь может управлять их раскрытием. Например, пользователь может иметь информацию входа в учетную запись банковского Интернет-счета в карте совокупностей банков и может защищать ее с помощью финансового ключа, который может иметь одну продолжительность действия использования. В таком случае он, вероятно, должен вводить в финансовый пароль каж-

дый раз, когда он может хотеть осуществлять доступ к веб-узлу банка посредством осуществления доступа к входу в учетную запись и паролю, сохраненному на банковской карте. В рамках каждой банковской карты, пароль веб-узла может создаваться случайно, чтобы максимизировать энтропию, и сохраняться для использования автовхода в учетную запись посредством NUTbook. Могут добавляться дополнительные уровни, но это зависит от сложности информации пользователя и того, сколько он может хотеть запоминать. Может быть предусмотрен главный пароль 13720, который может обходить некоторые или все иерархические пароли. Главный пароль может тщательно выбираться или случайно формироваться для максимальной защиты и может сохраняться в безопасном месте. С использованием этой технологии на основе иерархических паролей, пользователь, возможно, просто должен тщательно выбирать набор паролей, которые может быть трудно угадывать, но которые могут проще запоминаться пользователем, просто вследствие уменьшения числа паролей, которые он, возможно, должен запоминать, и это может формировать основу его SSO-доступа.

Фиг. 138 проходит через процесс ввода пароля для открытия Nut-контейнера 13806 персонального документа. Этот документ может защищаться только посредством основного ключа уровня, так что ввод основного пароля 13710 для того, чтобы осуществлять доступ к основному ключу, с тем чтобы аутентифицироваться в NUTserver 13804, может быть достаточным для того, чтобы открывать Nut-контейнер, хранящий персональный документ 13806. На фиг. 139 маршрут главного пароля 13720 может быть точно идентичным маршруту основного пароля.

Фиг. 140 показывает то, как может открываться рабочий документ, защищенный посредством рабочего пароля 13716 второго уровня. Основной пароль 13710 может предоставляться, чтобы осуществлять доступ к основному ключу, затем рабочий пароль может вводиться, чтобы получать доступ к ключу 14008 рабочего уровня, который может отмыкать Nut-контейнер 14010 рабочего документа. На фиг. 141 маршрут отмыкания главного пароля 13720 может оставаться идентичным маршруту, показанному на фиг. 139, он по-прежнему может представлять собой одноэтапный доступ, так что главные пароли могут создаваться более защищенным способом.

Фиг. 142 показывает более подробную схему кэша 13520 NUTbook-ключей. Он может иметь секцию, сегментированную для ключей, ассоциированных с NUTserver 14210, и он может иметь секцию, сегментированную для использования в различных совокупностях 14230 карт.

Фиг. 143 показывает блок-схему последовательности операций способа того, как NUTbook может просматривать каталожную карту.

Сохраняемое владение представляет собой концепт, который связан со смешиванием Nut-контейнеров различных владельцев. Предположим, что Alice получает новую работу в компании Acme, и они оба могут использовать приложения на основе NUTS, чтобы управлять маловажными деталями организации своих соответствующих контактов и/или цифровых ключей. Дополнительно, Acme может использовать Nut-контейнеры, чтобы управлять Nut-контейнерами доступа и тщательно замыкать корпоративные документы посредством уровня доступа отделом и/или сотрудниками. Когда Alice нанимается, отдел кадров Acme может выдавать Alice общий корпоративный Nut-контейнер доступа: он может представлять собой Nut-контейнер доступа, который может обеспечивать возможность Alice искать такую информацию, как внутренние корпоративные списки контактов, списки клиентов и/или различные корпоративные документы. Acme NUTS-системы могут индивидуально настраиваться и/или конфигурироваться, чтобы предоставлять доступ к чувствительным документам, которые могут сохраняться в Nut-контейнерах, посредством обертывания копии рабочих данных в обертывающий Nut-контейнер, замыкаемый посредством конкретного Nut-контейнера доступа сотрудника и корпоративного главного ключа. Владение (RAT) этих корпоративных Nut-контейнеров может всегда принадлежать Acme. Аналогично, персональные Nut-контейнеры Alice могут всегда иметь ее в качестве RAT. Способность четко задавать владельца криптографическим способом может обеспечивать возможность обработки каждого Nut-контейнера надлежащим образом каждым соответствующим владельцем в NUTS-окружениях. Эта сохраняемая характеристика владения Nut-контейнеров может обеспечивать возможность Alice смешивать свои Nut-контейнеры с Nut-контейнерами Acme на любом устройстве, которые она может использовать, и поддерживать управление ими. То же самое может применяться к Nut-контейнерам Acme на устройствах Alice. Как Alice, так и Acme могут задавать продолжительность действия своих соответствующих Nut-контейнеров доступа как относительно короткий период. Например, продолжительность действия может задаваться в 60 дней на Nut-контейнерах, сохраненных на внешних системах. Следовательно, каждые 60 дней ключи могут заменяться на новые посредством каждого владельца Nut-контейнеров, принадлежавших им, или они могут быть автоматически удалены посредством внешнего NUTserver, управляющего ими. Удаления могут возникать принудительно, если в соответствующие NUTserver могут отправляться команды удаления в соответствующем Nut-контейнере доступа, и он может кодироваться, чтобы систематически удалять некоторые или все затрагиваемые Nut-контейнеры владельца. В силу этого, каждая сторона может иметь способность поддерживать управление своими Nut-контейнерами во внешних системах прямо или косвенно. Таким образом, если Alice уходит на новую работу, она может знать, что ее персональная контактная информация касательно того, что она может оставлять копию на своем корпоративном рабочем столе, может автоматически удаляться через 60 дней или меньше. То же

может применяться для любых находящихся в собственности Nut-контейнеров Acme, оставшихся на персональных устройствах Alice: если отсутствует обновленный Nut-контейнер доступа, более нет ассоциированных Nut-контейнеров в системе. Этот тип смешивания Nut-контейнеров может быть предназначен, чтобы разрешать извечную проблему манипулирования двумя или более отдельными списками контактов и различными наборами мер обеспечения безопасности для взятия работы на дом. Теперь Alice может всегда использовать свой персональный NUTbook в качестве основного источника контактов в свой личной и профессиональной жизни, и она может обоснованно удостоверяться, что он может быть защищенным.

В другом варианте осуществления контактная NUTbook-карта может переносить ссылки на или встраивать внешние Nut-контейнеры, которые содержат персональную информацию для окружающих. Внешний Nut-контейнер от Bob может принадлежать не Alice, а Bob. Bob может отправлять Alice предварительно скомплектованный Nut-контейнер контакта с ограниченными сведениями о себе и может поддерживать свое владение в NUTS-окружении Alice. NUTbook-запись Alice для Bob может встраивать этот Nut-контейнер в запись контакта для Bob либо непосредственно, либо посредством ссылки. Каждый раз, когда Bob изменяет часть или всю информацию относительно себя, такую как новый почтовый адрес, новый рабочий адрес, телефонные номера или другая затрагиваемая информация, он может отправлять обновление своего предварительно скомплектованного Nut-контейнера контакта Alice посредством любого доступного средства, и как только NUTserver Alice распознает его, он может автоматически обновлять соответствующий встроенный внешний Nut-контейнер в карте для Bob в NUTbook Alice. Затем NUTbook Alice может выполнять контактное приложение для того, чтобы обрабатывать обновленную карту, которая может приводить к обновлению в карте Alice для Bob. Этот последний этап может гарантировать то, что запись в карте Alice для Bob никогда не может терять свою предысторию по информации Bob, и она может разыскивать различные статистические изменения информации Bob, когда она может хотеть этого. Некоторые или все эти этапы могут возникать автоматически без вмешательства на общепринятых, доверенных RBK-взаимосвязях. Это может означать то, что часть или все доверенные RBK-взаимосвязи Alice могут обновлять контактную информацию практически без вмешательства вручную, что может приводить к большой экономии по времени и усилиям для Alice и для каждого из ее друзей. Если Alice имеет 99 RBK-контактов, и могут возникать 50 обновлений, то только 50 изменений возможно должны инициироваться самими затронутыми людьми, и остальные могут обрабатываться автоматически посредством NUTserver каждого затрагиваемого пользователя. В традиционной настройке адресной книги, 50 обновлений могут становиться 50 обновлениями от затрагиваемого человека, 50 уведомлениями для 99 друзей, информирующими их в отношении изменения, причем каждый из 99 друзей выполняет вплоть до 50 обновлений в собственные адресные книги наряду с некоторым уровнем ошибок транскрипции в рамках почти 10000 событий, которые могут порождать 50 обновлений, уже не говоря об общем времени, потраченном посредством 100 человек, которые могут вовлекаться. Этот вариант осуществления может разрешаться альтернативно при наличии централизованной услуги, но такие услуги могут предоставлять ограниченную конфиденциальность, доступ, владение и/или управление. NUTS-решение может подчеркивать децентрализацию в максимально возможной степени при попытке поддерживать согласованно высокие уровни конфиденциальности, предыстории, журналов аудита и/или владения.

Услуги на основе NUTS

Услуги на основе NUTS могут расширять использование Nut-контейнеров на более глобальную сеть, такую как Интернет, так что Nut-контейнеры могут использоваться между несколькими удаленными сторонами. Таблица на фиг. 144 перечисляет примеры различных веб-услуг, которые NUTS может поддерживать и предлагать, и фиг. 145 показывает упрощенную схему размещения сети для этих услуг. Некоторые или все услуги могут предлагать многоуровневые комплекты услуг с самыми нижними уровнями, предлагаемыми бесплатно с ограничениями. Платежи для более высокоуровневых комплектов могут быть произведены непосредственно или анонимно через отдельно приобретаемые ваучеры на льготное предоставление услуг. Некоторые или все услуги могут использоваться анонимно в различные степени.

Услуги на основе NUTS: NUTmail

NUTmail-сервер, проиллюстрированный на фиг. 146, показывает почтовую веб-услугу, которая передает некоторые или все свои сообщения через Nut-контейнеры между своими зарегистрированными пользователями. Кроме того, он может поддерживать авторегистрации, анонимные регистрации, анонимные каналы и/или связь на основе RBK. Сервер может взаимодействовать с NUTbook- и/или NUTserver-приложениями. NUTmail может иметь клиентский компонент, который может выполняться на устройстве пользователя, чтобы обеспечивать возможность ему управлять, редактировать, отображать, составлять, отправлять и/или принимать почтовые сообщения.

Фиг. 147 показывает процесс для установления анонимной регистрации на NUTmail-сервере автоматизированным способом. Пользователь может обеспечивать контакт 14700 сервера с предварительно скомплектованным Nut-контейнером, который может содержать предпочтительный уже существующий контактный способ, такой как, но не только, адрес электронной почты, телефонный номер с поддержкой

текста и/или веб-браузер. Сервер может подтверждать 14702 запрос и может отправлять 14704 запрос пользователю с использованием предпочтительного способа контакта. Пользователь может вводить требуемую информацию из запроса, и сервер может создавать случайно сформированный идентификатор для входа в систему и пароль, который может использовать максимальную энтропию способа шифрования на 14706. Сервер также может формировать RBK-пару с пользователем, которая может использоваться в части или всей связи между пользователем и сервером/администратором. Пользователь может сохранять учетные данные для входа в учетную запись и RBK-пару в своей NUTbook в карте для собственной контактной информации 14708. Таким образом, пользователь может анонимно регистрироваться на NUTmail-сервере преимущественно автоматизированным способом 14710.

Идентификатор для входа в систему и RBK, которые могут создаваться во время процесса регистрации, могут использоваться пользователем только для того, чтобы обмениваться данными с NUTmail-сервером; таким образом, он может считаться закрытым каналом между пользователем и сервером. Когда пользователь хочет обмениваться данными с другим пользователем, который также может использовать NUTmail, канал связи, возможно, должен устанавливаться с этим пользователем на NUTmail-сервере, как проиллюстрировано на фиг. 148. Канал связи может содержать пару случайно сформированных дополнительных адресов электронной почты, которые могут присоединяться к зарегистрированным учетным записям каждого пользователя в качестве псевдонимов. NUTmail-сервер может не отслеживать эти пары псевдонимов, как только канал связи может устанавливаться и верифицироваться, чтобы лучше сохранять анонимность взаимосвязей. Эти псевдонимы могут быть аналогичными по функции RBK в том, что они могут использоваться только двумя участниками канала. Случайный характер формирования псевдонимов может не выдавать подсказки для идентификаторов участников во время транзита почтовых сообщений в Интернете. Непосредственно контент электронной почты самостоятельно может быть вмонтирован в Nut-контейнер, защищенный посредством RBK-способов, дополнительно защищающих рабочие данные. Это позволяет предоставлять два отдельных уровня способов на основе взаимосвязей и запутываний, которые могут минимизировать часть или весь нежелательный спам и/или сниффинг сторонней электронной почты. После того, как канал связи может надлежащим образом устанавливаться, далее обмен почтовыми сообщениями может быть довольно стандартным, как показано на фиг. 149.

Объяснение безопасности для NUTmail-сервера может обобщаться следующим образом:

Анонимные регистрации могут означать, что скомпрометированный сервер может раскрывать очень мало информации относительно зарегистрированных пользователей и/или контента их электронной почты.

Инкапсуляция почтовых сообщений в RBK-зашифрованных Nut-контейнерах может предоставлять другой независимый уровень безопасности контента. Взломанные серверы могут раскрывать только сообщения, защищенные посредством Nut-контейнеров.

NUTmail-каналы связи с использованием пар псевдонимов могут запутывать метаданные электронной почты.

Сервер может сохранять данные спаривания псевдонимов не постоянно, а только достаточно долго для верификации канала.

Сервер может сохранять почтовые сообщения в течение очень короткого периода времени. Он может быть конфигурируемым пользователем, но по умолчанию может быть предусмотрено то, что сообщения могут быть вычеркнуты после того, как он может принимать информацию из NUTmail-клиента или NUTserver пользователя касательно того, что по меньшей мере 2 копии могут существовать вне рамок сервера или после предварительно сконфигурированной длительности.

Краткая история почтовых сообщений может обеспечивать возможность серверу иметь очень нестрогие требования по долговременному хранению данных.

Случайно сформированные входы в учетную запись, псевдонимы, пароли и/или RBK могут полностью использовать доступную энтропию данных, что может приводить к дополнительной защите.

Может быть непросто использовать NUTmail-сервер без интегрированного упрощения NUTbook, хотя это может быть возможным. Идентификатор для входа в систему, пароль и/или псевдонимы могут формироваться с использованием способов на основе максимальной энтропии и могут выглядеть как беспорядочная смесь длинной строки случайных символов. Может быть предусмотрено соответствие 1:1 между взаимосвязью и парой псевдонимов, так что число псевдонимов, которые, вероятно, придется отслеживать пользователю, может становиться многочисленным очень быстро. Преимущество этой технологии связи может заключаться в том, что данные, сформированные посредством участников, могут быть бесполезными сами по себе, и некоторый смысл может извлекаться только через технологии целевого наблюдения данных и/или сложного реконструирования.

Требования по объему хранения данных NUTmail-сервера могут отличаться от обычного почтового сервера: он может использовать намного меньше пространства в расчете на пользователя на постоянной основе. Когда NUTserver или NUTmail-клиент пользователя может указывать то, что по меньшей мере две копии электронной почты могут существовать за пределами NUTmail-сервера, NUTmail-сервер может удалять Nut-контейнер этой электронной почты безвозвратно. Этот тип простого правила может

обеспечивать возможность каждому участнику канала устанавливать минимум по две или более копий коммюнике. NUTmail-сервер может использовать NUTserver каждого зарегистрированного клиента, чтобы разгружать долговременное хранение данных в максимально возможной степени, за счет этого уменьшая собственные текущие требования по хранению в расчете на пользователя. NUTmail-сервер может иметь только новые почтовые сообщения для зарегистрированных пользователей, поскольку каждый пользователь может загружать и реплицировать предыдущие электронные письма на собственных NUTchat-клиентской/NUTserver-системах.

Услуги на основе NUTS: NUTchat

NUTchat может представлять собой анонимную чат-услугу на основе Nut-контейнеров. Она может предлагать следующие признаки чата.

Она может поддерживать анонимную регистрацию, попарные случайные псевдонимы и/или RBK.

Она может иметь возможность предоставлять локальные номера телефона концентратора NUTchat для анонимности.

Она может поддерживать одновременные чаты для сотовых телефонов и не для сотовых телефонов.

Она может поддерживать сеансы чата на основе SMS/MMS и Интернета одновременно.

Она может поддерживать функции предыстории, аналогичные функциям предыстории NUTmail-сервера.

Предыстория чатов может сохраняться в рамках каждого хранилища записей контактов либо она может сохраняться в Nut-контейнере, и к ней можно обращаться посредством целевой записи контакта, а не посредством просто телефонных номеров или адресов чата.

Предыстория чатов может постоянно сохраняться для личного использования без необходимости услуги NUTchat.

NUTchat может представлять собой специализированную услугу для сообщений чата, которые могут содержаться в Nut-контейнере.

Случайно сформированные входы в учетную запись, псевдонимы, пароли и/или RBK могут полностью использовать доступную энтропию данных, которая может приводить к дополнительной защите.

Она может мультиплексировать маршруты связи, чтобы обеспечивать доставку сообщений и показывать виртуальные сеансы чата.

Пример схемы сети показан для NUTchat-сервера на фиг. 150. Его процедуры регистрации могут быть аналогичными способам, используемым посредством NUTmail-серверов, и могут предлагать некоторые или все анонимные признаки для его пользователей. NUTchat-клиент на основе Nut-контейнеров может выполняться на пользовательских устройствах, и базовая конфигурация потоков данных показана для сеансов чата между тремя участниками на фиг. 151. Она может представлять собой стандартную топологию пересылки текстовых сообщений с NUTchat-сервером, выступающим в качестве координатора в середине 15100. Поскольку NUTchat может быть основан на Nut-контейнерах, вся предыстория чатов сеанса может сохраняться в Nut-контейнере и в силу этого может использовать преимущество признаков репликации/распространения/синхронизации NUTserver автоматически в случае надлежащего конфигурирования. NUTserver может быть выполнен с возможностью приоритезировать Nut-контейнеры NUTchat таким образом, что они могут обрабатываться более своевременно вследствие характера интерактивности в реальном времени в сеансе чата. Внимательное рассмотрение на фиг. 151 показывает то, что идентичные Nut-контейнеры чата существуют в нескольких местоположениях; оно показывает то, что топология чата может быть эквивалентной оптимизированной синхронизации состояний данных во множестве физических местоположений. Фиг. 152 представляет собой пример потоков данных процесса, который может реплицировать NUTchat-сеансы с использованием NUTserver пользователя. Поскольку каждый участник чата может сохранять часть или всю предысторию сеансов чата в Nut-контейнере 15122-15126, NUTserver 15238 может распространять изменения этих Nut-контейнеров через свое равноправные NUTserver, такие как 15242. Посредством надлежащей синхронизации данных этим методическим способом, когда пользователь загружает NUTchat-клиент 15260 на свое устройство #4 15240, он может видеть предысторию сеанса, идентичную предыстории сеанса, которую он может оставлять на устройстве #2, и NUTchat-сервер никоим образом не вовлечен в актуализацию его устройства #4. Когда сеанс чата инициируется, и когда анализ Nut-контейнеров чата на любой стороне канала посредством соответствующих NUTchat-клиентов может определять то, что он является несинхронизированным, то принудительная процедура синхронизации может автоматически инициироваться, чтобы обновлять сеанс на последнюю версию (следует отметить, что классификация предыстории чатов может рассматриваться по существу в качестве более нового состояния рабочих данных, иначе Nut-предыстории). Например, Alice может иметь долгосрочный анонимный NUTchat-канал с Bob, но каким-то образом она может терять или удалять свой Nut-контейнер чата, сохраняющий эту предысторию сеанса, на своем смартфоне. Когда она возобновляет этот NUTchat-сеанс с Bob и может вступать в контакт через NUTchat-сервер, сервер может принимать номера версий сеанса как от Alice, так и от Bob, и они могут показывать то, что Bob может иметь более позднюю версию сеанса, чем Alice. В этот момент, копия Nut-контейнера чата Bob может запрашиваться автоматически и может отправляться Alice через NUTchat-сервер, и NUTchat-клиент Alice может подтверждать предысторию сеанса Bob как собственную, и сеанс чата может про-

должаться с общим видом его предыстории и в силу этого его контекста. Может использоваться очень небольшой объем хранилища в этом сценарии посредством NUTchat-сервера, и часть или вся информация сеанса может сохраняться конечными пользователями под их управлением. После того, как версии сеанса чата могут синхронизироваться, сообщения чата, отправленные друг другу после этого, могут содержаться в Nut-контейнерах, хранящих только новое сообщение чата в сеансе, а не всю предысторию, и NUTchat-клиенты на каждом конце могут отвечать за обновление своего кумулятивного сеанса чата, соответственно, в силу чего они могут уменьшать размер переносов данных в текущем сеансе чата.

Кроме того, NUTbook Alice может задавать ссылки в своей записи контакта для Bob, с тем чтобы ссылаться или указывать на Nut-контейнеры чата и Nut-контейнеры электронной почты таким образом, что часть или вся релевантная статистическая связь с Bob может индексироваться согласно информации Bob, что может обуславливать систематическое сопоставление контекста во взаимосвязи, сохраненной под управлением Alice.

NUTchat-клиенты могут участвовать в диалоге, который может предусматривать агностические к тракту сеансы чата для надежности, избыточности и/или запутывания. Фиг. 153 показывает типичный шаблон потока данных для трех отдельных сеансов чата между Bob и Alice, который может использовать максимум три различных чат-услуги и/или идентификатора чатов. Иногда, этот тип разделения и сегрегации может быть желательным и удобным для сторон, которые могут участвовать. В другие моменты времени, он может навязываться пользователю в силу вариантов выбора, осуществляемых другим участником: например, Bob может хотеть только учетную запись в чат-услуге B, так что Alice можно командовать создавать вход в учетную запись в услуге B, чтобы проводить чат с Bob. Тем не менее, до такой степени, что NUTchat-клиент может взаимодействовать с другими чат-услугами, он может обеспечивать возможность агломерирования нескольких отдельных сеансов чата между идентичными двумя пользователями в агностический к тракту сеанс чата, как показано на фиг. 154, который может называться диалогом. Nut-контейнеры чата могут представлять сбой базовую среду сообщений таким образом, что некоторые или все могут иметь номера версий, и копия Nut-контейнера может отправляться по некоторым или всем трем трактам сеанса чата одновременно. Независимо от того, какой Nut-контейнер чата может связываться с другим NUTchat-клиентом первым, он может обрабатываться, а другие игнорироваться (или могут объединяться посредством объединения NUTserver Nut-контейнеров и затем отбрасываться). Иногда вследствие характера транспортных ограничений, Nut-контейнеры чата могут преобразовываться в точные, защищенные текстовые сообщения, подходящие для транспортной платформы. В этом способе, разговор может сохраняться по нескольким путям, и только самая актуальная версия может вообще показываться каждому участнику, и процесс может основываться не на функциональности хранения и/или организации отдельных поставщиков чат-услуг, только на их транспортных механизмах. Избыточные пути могут минимизировать или фактически исключать сбои при транспортировке для диалога. Предыстория, которую может сохранять каждая транспортная услуга, может быть бесполезной, поскольку она может защищаться посредством Nut-контейнера в расчете на каждое сообщение, в силу чего контент может быть непрозрачным. Транспортные механизмы могут представлять собой любой канал, который может обеспечивать возможность Nut-контейнерам пересылаться, такой как, но не только, почтовые серверы, FTP-серверы, сетевые файловые системы, соединения "точка-точка", Wi-Fi-протоколы, Bluetooth-протоколы и/или любой другой способ цифровой передачи. Свойства синхронизации Nut-контейнера могут предоставлять возможность вовлечения сеансов чата только посредством использования совместно используемого Nut-контейнера, выполненного с возможностью иметь по меньшей мере двух писателей и общепринятый способ для пользователей, чтобы осуществлять доступ к Nut-контейнеру. Этот вариант осуществления может показывать то, насколько относительно простым может быть освобождение функциональности чат-систем при защите пользовательских данных независимо от услуги и повышение общей надежности механизмов передачи пользователем.

Услуги на основе NUTS: NUTcloud

NUTcloud может представлять собой сервер хранения данных на основе Интернета, доступный для любого NUTS-пользователя, как проиллюстрировано на фиг. 155. NUTcloud может поддерживать анонимную регистрацию, попарные случайные псевдонимы и/или RBK. Он может эффективно интегрироваться с персональными NUTserver, чтобы увеличивать радиус действия и доступность персональной NUTS-сети. NUTcloud может сохранять Nut-контейнеры, и его ограничения по объему хранения и по полосе пропускания могут затрагиваться посредством уровней уровня предоставления услуг и конфигурируемых пользователем политик. Учетные NUTcloud-записи могут взаимодействовать с другими услугами на основе NUTS, чтобы предоставлять дополнительное постоянное и/или доступное хранилище: т.е. он может резервировать NUTmail- и/или NUTchat-сообщения.

На базовом уровне предоставления услуг, он может предлагать достаточный уровень объема хранения и полосы пропускания для общего личного использования. Его основная цель может состоять в том, чтобы упрощать осуществление доступа данных, сохраненных в Nut-контейнерах, из любой точки доступа в Интернете. Он может эффективно интегрироваться с NUTserver, чтобы синхронизировать некоторые или все данные Alice дома и в дороге.

NUTcloud в сочетании с персональным NUTserver может предлагать идентичный или лучший уро-

вень синхронизации по сравнению с любой централизованно управляемой облачной услугой на основе Интернета; тем не менее, в отличие от популярных услуг облачной синхронизации в свободном доступе, NUTcloud может предлагать полную анонимность, управляемую пользователем конфиденциальность, полную предысторию, полный журнал аудита и/или защищенное владение данными.

Услуги на основе NUTS: NUTnet

NUTnet может представлять собой веб-сервер на основе Nut-контейнеров, доступный для NUTS-пользователя, как проиллюстрировано на фиг. 156. NUTnet может поддерживать анонимную регистрацию, попарные случайные псевдонимы и/или RBK. NUTnet может сохранять Nut-контейнеры, и его ограничения по объему хранения и по полосе пропускания могут затрагиваться посредством уровней уровня услуги и конфигурируемых пользователем настроек политики. Учетные NUTnet-записи могут взаимодействовать с другими услугами на основе NUTS, чтобы осуществлять доступ к большему объему постоянного и/или доступного хранилища: например, он может осуществлять выборку Nut-контейнеров из NUTcloud и/или NUTserver.

Совместное использование контента веб-страниц, сохраненного в Nut-контейнерах, может обеспечивать возможность пользователям управлять тем, кто может просматривать контент, и это может осуществляться на криптографическом уровне. Пользователь может иметь RBK-пару с владельцем контента, чтобы просматривать отправленные страницы. Можно сказать, что она может представлять собой антисоциальную социальную сеть, частную социальную сеть и/или аутентифицированную социальную сеть. Ничего из контента не может добываться посредством NUTnet-сервера или другой неавторизованной третьей стороны, поскольку он не может иметь ни одного из ключей для контента. При условии, что контент может сохраняться и защищаться в Nut-контейнерах, владелец может сохранять управление над ним. Владелец также может просматривать часть или всю предысторию, ассоциированную с ее сообщениями, в локальном Nut-хранилище, если оно также сконфигурировано с возможностью реплицировать и синхронизировать Nut-контейнеры локально. Могут возникать случаи, когда пользователь чувствует, что совместное использование фотографий и видео близкими друзьями и семьей может быть личным делом, и что третья сторона не может иметь право владеть их копией для своего использования без знания и/или разрешения отправителя. NUTnet может создаваться для этих ситуаций, требующих конфиденциальности в группе пользователей.

Профессиональные фотографы могут устанавливать закрытые веб-страницы для потенциальных клиентов, чтобы просматривать фотографии, с защитой авторских прав, с огромным количеством подробностей и управлением тем, кому могут выдаваться ключи и на сколько. Nut-контейнеры веб-страниц могут регистрировать часть или всю активность по фотографиям, чтобы создавать журнал аудита для фотографа. Руководители проекта могут устанавливать закрытые веб-страницы для координирования активности между членами проекта. С точки зрения безопасности, процесс регистрации может быть необязательным вследствие средств управления доступом, встроенных в Nut-контейнер, но он может служить в качестве функции организации и разделения в NUTnet-сервере.

Услуги на основе NUTS: NUThub

В настоящее время, может не быть универсально подтвержденного стандарта в отношении того, как Интернет вещей (IoT) может обмениваться данными и/или функционировать. IoT может быть растущей областью аппаратных продуктов, которые могут иметь встроенные возможности организации сетей и могут обеспечивать возможность пользователям управлять и отслеживать функции продукта удаленно от различных персональных вычислительных устройств. Множество IoT-продуктов могут отправлять постоянный поток данных из своих датчиков обратно промышленному производителю для сбора и анализа, иногда без ведома пользователя-владельца продукта. Рабочий режим некоторых или всех этих IoT-устройств может поднимать много вопросов вторжения в личную жизнь на основе их диапазона и способов сбора данных, поскольку продукты могут быть предназначены для большинства закрытых областей дома пользователя. IoT-инфраструктуры, которые получают некоторое использование, могут предоставляться посредством поставщиков IoT-оборудования для их семейства продуктов. NUThub может представлять собой услугу перенаправления пакетов, чтобы упрощать обработку сообщений на основе NUTS, которые могут создаваться посредством NUTS-совместимых IoT-устройств, называемых Интернетом NUTS (IoN). Как проиллюстрировано на схема сети на фиг. 157, IoN может представлять собой стандарт на основе NUTS для обмена данными защищенно и закрыто со своими IoN-совместимыми устройствами дома. Самый нижний уровень обслуживания в NUThub может быть доступен для любого, кто может иметь зарегистрированную учетную запись в любой услуге на основе NUTS. Учетная запись может быть анонимной. NUThub может работать с Nut-контейнерами, и он может ставить в очередь определенное число сообщений. NUThub может взаимодействовать прозрачно с NUTcloud и/или NUTserver, чтобы осуществлять доступ к дополнительному хранилищу.

NUThub-топология может быть выполнена с возможностью работать несколькими способами. На фиг. 158 показана прямая топология, в которой каждое IoN-устройство у пользователя дома может устанавливать независимые связи с IoN-серверами 15804 производителя, NUThub 15802 и/или пользовательскими устройствами 15806, 15822 и 15824 управления. Эта топология может обеспечивать возможность производителям иметь более прямой доступ к устройствам в вашем доме, и пользователь может фильт-

ровать исходящие Nut-пакеты только вплоть до характеристик фильтрации каждого устройства: он может представлять собой преобладающий способ связи, используемый посредством IoT-устройств сегодня.

Предпочтительная NUThub-топология может быть косвенной, как проиллюстрировано на фиг. 159. Некоторые или все IoN-устройства могут обмениваться данными через обозначенный NUTserver-концентратор 15930 перед выходом из LAN 15920 и последующим обходом NUThub 15902. Эта топология может предоставлять возможность подстройки правил фильтрации для IoN-сообщений, выходящих из дома Alice, на основе ее уровня комфорта. Устройство 15930 NUTserver-концентратора может содержать настольный PC, прибор специального назначения или даже составлять часть Wi-Fi-маршрутизатора 15920. Если обозначенный NUTserver-концентратор 15930 выключен или недоступен, IoN-устройство не может обмениваться данными с внешним миром.

Конфигурация NUTserver-концентратора показана на фиг. 160. В знакомом NUTserver 15930, может быть предусмотрен компонент, называемый NUThub/IoN-интерфейсом 16032. Этот модуль может отвечать за обмен данными с NUThub 15902, IoN-устройствами 15922 и/или другими NUTserver-концентраторами 16052.

Интерфейсный модуль 16032 может регистрировать, ставить в очередь, перенаправлять, ретранслировать, обрабатывать и/или фильтровать IoN Nut-сообщения и из IoN-приборов и из IoN-устройств управления.

Более подробный вид NUThub/IoN-интерфейса показан посредством фиг. 161. Интерфейс 16032 может содержать некоторые или все эти семь функций или другие дополнительные функции. Индекс 16112 IoN-устройств может отслеживать некоторые или все IoN-устройства, зарегистрированные пользователем. Аутентификация 16114 IoN-устройств может аутентифицировать и может шифровать сообщения в/из IoN-устройств. Интерфейс может отслеживать фильтры и правила 16116 обработки сообщений пользователя.

Регистратор 16118 сообщений может регистрировать некоторые или все IoN-сообщения в долговременное хранилище. Очередь 16120 сообщений может временно хранить недоставленные сообщения. Кэш 16122 ключей устройства может сохранять некоторые или все ключи доступа для аутентификации и шифрования IoN-сообщений, и он может быть осуществлен в защищенных аппаратных средствах запоминающего устройства при наличии. Удаленный управляющий интерфейс 16124 может представлять собой модуль, который может предоставлять возможность удаленной активации конкретных функций IoN-устройства.

Более подробный вид NUThub/NUTserver/IoT-интерфейса на любом IoN-устройстве показан посредством фиг. 162. Интерфейс 16210 может содержать некоторые или все эти семь функций или другие дополнительные функции. Индекс 16212NUT-контейнеров может отслеживать некоторые или все Nut-контейнеры, сохраненные на устройстве, релевантные для администрирования и управления IoN-устройствами. Модуль 16214 аутентификации может аутентифицировать и может шифровать сообщения в и/или из устройства в производителя, NUThub и/или NUTserver-концентратор. Интерфейс может отслеживать фильтры и правила 16216 обработки сообщений пользователя. Регистратор 16218 сообщений может регистрировать некоторые или все IoN-сообщения в долговременном хранилище. Очередь 16220 сообщений может временно хранить недоставленные сообщения. Кэш 16222 ключей устройства может сохранять некоторые или все ключи доступа для аутентификации и шифрования IoN-сообщений, и он может быть осуществлен в защищенных аппаратных средствах запоминающего устройства при наличии. Удаленный управляющий интерфейс 16224 может представлять собой модуль, который может предоставлять возможность удаленной активации конкретных функций IoN-устройства. IoN-устройство может иметь ограниченный набор функциональности для настраиваемой фильтрации вследствие своих аппаратных ограничений. Оно также может иметь ограничения по объему хранения, которые могут ограничивать число сообщений, которые он может регистрировать и ставить в очередь. Следовательно, если предыстория и журналы аудита могут быть важными, пользователю можно настоятельно рекомендовать использовать косвенную IoN-топологию, как проиллюстрировано на фиг. 159, что может обеспечивать ему возможность осуществлять доступ к улучшенным функциональностям, которые могут предоставляться посредством NUTserver-концентратора. Этот интерфейс 15922 не ограничен конкретными для IoN/IoT устройствами, любое вычислительное устройство может иметь аналогичный интерфейс, если разработчик может создавать интерфейс для него, и придерживается рабочих режимов IoN-устройства; дополнительно, любое устройство, которое может иметь версию NUTserver, выполняющегося на нем, может допускать выступание в качестве самого IoN-устройства.

Когда Alice покупает новое IoN-устройство, она, возможно, должна добавлять его в свою сеть и конфигурировать его. Блок-схема последовательности операций способа на фиг. 163 показывает этапы, которые может осуществлять Alice для того, чтобы надлежащим образом регистрировать свое новое IoN-устройство в своей сети на основе NUTS. Способ конфигурирования IoN-устройства может заключаться в том, чтобы устанавливать RBK-взаимосвязь с ним через NUTbook Alice. Этапы 16302 и 16304 могут обеспечивать возможность NUTserver-концентратору ретранслировать конкретную для устройства информацию в ее NUTbook, и в свою очередь NUTbook может создавать каталожную карту IoN/IoT-

устройства, заполнять модель, версию и/или порядковые номера, формировать RBK-пары и отправлять ее обратно в IoN-устройство через NUTserver-концентратор. Этап создания каталожной карты для IoN-устройства может создавать Nut-контейнер, который может создавать Nut-идентификатор для этого Nut-контейнера; в силу этого IoN-устройство может далее быть отпечатано с Nut-идентификатором его Nut-контейнера каталожных карт. Этот этап может быть аналогичным выбору IP-адреса для нового устройства в собственной домашней сети, но потенциальные преимущества использования Nut-идентификатора могут быть далеко идущими. Назначенный Nut-идентификатор для IoN-устройства также может служить в качестве постоянного способа ссылаться на устройство независимо от его фактического IP-адреса и/или местоположения.

IoN-устройство может сбрасываться до заводского режима таким образом, что новый Nut-идентификатор может быть отпечатан на нем посредством нового или идентичного владельца.

После того, как каталожная IoN-карта сохраняется в NUTbook Alice, процесс конфигурирования может переходить к этапу 16306, и он может проверять то, могут или нет быть предусмотрены MIO-компоненты, необходимые для того, чтобы расшифровывать конфигурационные данные устройства, отображать их и/или задавать их. После того, как надлежащие настройки установлены на конфигурационных экранах, Alice может сохранять настройку в свою каталожную IoN-карту для устройства и может отправлять ее в интерфейс NUTserver-концентратора для отправки в IoN-устройство 16314. Устройство может принимать конфигурационный Nut-контейнер, может аутентифицировать его, может декодировать его, может проверять достоверность его, затем может применять изменения его внутренней системы. После завершения, оно может отправлять Nut-контейнер обратно в NUTserver-концентратор, указывающий его состояние. Alice может отслеживать это устройство, и она может видеть сообщения из него автоматически.

IoN-устройства могут работать в режиме, в котором некоторые или все сообщения могут представлять собой Nut-контейнеры, и в силу этого могут быть предоставлены идентичный уровень конфиденциальности и управление Nut-контейнерами по умолчанию. Поскольку Nut-контейнеры могут использовать MIO-компоненты, программные конфигурации, обновления микропрограммного обеспечения и/или программного обеспечения в устройства могут отправляться через идентичные MIOR-механизмы, и потенциал для устаревания может быть низким. NUThub может быть выполнен с возможностью, может гарантировать пользователю, что все может отслеживаться, регистрироваться и/или управляться им при необходимости, и что часть или вся исходящая информация, которая может собираться посредством IoN-устройства, могут фильтроваться, чтобы удовлетворять предпочтения конфиденциальности пользователя. В этом варианте осуществления, базовая NUTS-философия может распространяться на физические устройства таким образом, что устройство, которым владеет пользователь, может находиться под его управлением в некоторых или всех случаях, и некоторые или все данные, которые оно может формировать, также могут быть его. Сила MIO и его функциональностей может быть очевидной в этом сценарии, поскольку любой формат данных с надлежащим MIO-компонентом может быть проверен пользователем, в отличие от множества собственных протоколов.

Это ведет к важному модулю, называемому удаленным управляющим интерфейсом, показанным в 16124 и 16224. Он может представлять собой способ, посредством которого пользователь или производитель может разговаривать с IoN/IoT-устройством и могут инструктировать ему воздействовать на команды удаленно, которые называются в качестве Nut-контейнеров команд. RBK-аутентифицированные Nut-контейнеры команд могут обрабатываться, и владелец устройства (RAT) может выполнять любую команду, доступную для него. Это требование по аутентификации может обеспечивать возможность пользователю полностью управлять своей взаимосвязью с производителем посредством регулирования прав доступа производителя. Пользователь может обеспечивать возможность производителю устройства иметь полный доступ к нему, его поднабору и/или не иметь доступа. Это может предотвращать неавторизованный доступ к домашней сети Alice с использованием IoN/IoT-устройств в качестве точек входа: каждая точка IoN/IoT-доступа может теперь усиливаться посредством безопасности на основе NUTS. Поскольку упомянут обширный характер того, как Nut-контейнеры могут распространяться и могут отправляться вдоль сети intranet и/или Интернета, по существу Nut-контейнер IoN-команд может отправляться из любого места, где может быть предусмотрен надлежащий маршрут в IoN-устройство. Блок-схема последовательности операций способа на фиг. 164 показывает то, как удаленный управляющий интерфейс может обрабатывать Nut-контейнеры команд.

Характер NUThub и его удаленного управляющего интерфейса может обуславливать способность Alice полностью управлять некоторыми или всеми ее NUTS-совместимыми устройствами из любого места, где может быть предусмотрено подключение. Это может представлять защищенный протокол, посредством которого настраиваемые сообщения могут отправляться при управлении посредством взаимосвязей NUTbook Alice, представленных посредством RBK-пар. Он может представлять централизованный вид для Alice для всех ее IoN-устройств, но он может устанавливаться, конфигурироваться и/или поддерживаться децентрализованным способом. Если Alice управляет своими Nut-контейнерами, она может управлять некоторыми или всеми своими устройствами. Это может быть другой причиной, по которой когда Alice может решать использовать SSO-поддержку NUTS, она должна выбирать свои фра-

зовые пароли очень тщательно или использовать аппаратный ключ. В таких вариантах осуществления, роль производителя может быть урезана до роли производителя оборудования, а не роли нежелательного удаленного администратора персонального устройства, которое принадлежит Alice, и может располагаться в закрытой области дома Alice. Безопасность NUTS-окружения может представлять более унифицированный, усиленный и/или управляемый пользователем барьер, чем современные протоколы IoT, которые могут смещаться к предпочтениям и/или преимуществам изготовителя (разработчика).

Услуги на основе NUTS: NUTS-сервер сертификации Поскольку целостность NUTserver-процессов и протоколов может быть важна для доверия, что она может демонстрировать ожидаемое поведение, может быть предусмотрен NUTS-сервер сертификации (NCS), чтобы проверять достоверность NUTserver-установок на постоянной основе. Как показано на фиг. 165, NCS может быть доступна для любого NUTS-пользователя и может поддерживать анонимную регистрацию, попарные случайные псевдонимы и/или RBK. Она может иметь многоуровневый уровень обслуживания, при этом самый верхний уровень представляет собой официальную сертификацию посредством NCS-компании как "NUTS-сертифицированный". Основные функции NCS могут заключаться в том, чтобы отслеживать NUTserver на предмет надлежащего удаления Nut-контейнеров и/или обнаруживать неавторизованную подделку NUTserver-протоколов, поведений и/или процессов. Поскольку умные программисты могут идентифицировать тестовые сообщения и могут обходить их, архитектура того, как работают анонимные регистрации, может обеспечивать возможность тестовым NCS-сообщениям в NUTserver быть фактически необнаруживаемыми. Он может представлять собой преднамеренный уровень обслуживания, который пользователь может выбирать активировать на своем NUTserver. Могут быть предусмотрены автоматизированные процедуры, инициируемые посредством NCS, чтобы вводить целевой NUTserver с тестовыми Nut-контейнерами и обнаруживать то, могут или нет определенные действия применяться к ним согласно NUTserver-протоколам. На более высоких уровнях предоставления услуг, активное участие посредством тестеров может обеспечивать возможность еще более полных оценок в отношении состояния удаленного NUTserver.

Производители могут подписываться на тестирование уровня NUTS-сертификации постоянно поддерживать уровень NUTserver-соответствия, которое может быть сообщено их клиентуре, и гарантировать им, что их Nut-контейнеры могут обрабатываться соответственно. Процесс тестирования также может подчеркивать все неавторизованные модификации NUTS-окружений клиента, без ведома клиенту. С клиентской стороны, любой производитель, который может использовать NUTS-системы и технологии, но может не быть "NUTS-сертифицированным", может требовать большего числа запросов в отношении своих политик для обработки Nut-контейнеров. Пользователи могут конфигурировать свои NUTserver и/или NUTbook с возможностью взаимодействовать с таблицей поиска в общедоступных NCS-базах данных, чтобы оценивать свое состояние сертификации либо его отсутствие до взаимодействия с онлайновым производителем.

На фиг. 166 NCS 16520 может выполнять функции, которые могут обеспечивать ему возможность оценивать поведение удаленных NUTserver 16620-16624 производителя (или персональных NUTserver). Тестирование 16602 сообщениями на целостность по истечению срока может представлять собой способ, в котором Nut-контейнеры могут вводиться 16604 в систему и могут тестироваться сообщениями посредством удаленного управляющего интерфейса 16610 для существования в этой системе после времени истечения срока.

Например, если истекшие Nut-контейнеры содержатся на удаленном NUTserver, NUTserver может быть за рамками соответствия и может не быть "NUTS-сертифицированным". Длительные тесты 16608 на введение могут тестировать NUTserver в течение большего количества времени и на постоянной основе. Анализ и сертификация 16606 результатов может оценивать прилипание удаленных NUTserver к различным тестам на введение и может классифицировать NUTserver-установку. Проверка версий установленных NUTserver и версий исправлений может составлять неотъемлемую часть проверки того, что NUTserver могут быть обновленными и поддерживают соответствие. Давно устаревшая версия может указывать нестрогое обслуживание NUTS-протоколов обеспечения безопасности, и/или могут выполняться неавторизованные настраиваемые модификации, в силу чего приспособления могут быть медленнее. Тестирование также может включать в себя, но не только, проверку хеш-подписей различных чувствительных сегментов двоичного кода и/или введение из анонимных Интернет-адресов. Анонимная регистрация NUTserver в NCS-услуге может гарантировать то, что RBK могут задаваться для более глубокого тестирования более защищенным способом.

NCS не может гарантировать то, что NUTserver не может быть скомпрометирован, поскольку с достаточными знаниями и ресурсами любой пользователь или группа может в конечном счете обходить тестирование посредством NCS. Локальные проверки могут приводить к верхним уровням NUTS-сертификации. Для среднестатистического пользователя, отказ от взаимодействия с коммерческими NUTserver, которые не могут подтверждаться на самых верхних уровнях, может представлять собой хорошую политику. Для взаимодействия с персональными NUTserver, базовый уровень автоматического свободного тестирования из NCS может быть минимальным требованием до взаимодействия с ним.

Организация сетей на основе NUTS для Wi-Fi/Ethernet-маршрутизатора

Фиг. 167 показывает вариант осуществления схемы размещения сети для персонального Wi-Fi/Ethernet-маршрутизатора 16710 на основе NUTS. Маршрутизатор может работать с использованием нормальных протоколов, которые могут быть предусмотрены в Wi-Fi-связи, а также использовать обмен сообщениями на основе NUTS в качестве альтернативного протокола. NUTS Wi-Fi-маршрутизатор может устанавливаться и конфигурироваться как любое IoN-устройство, за счет чего владелец устанавливает RBK-взаимосвязь с ним и может сохранять информацию в свою каталожную IoN-карту через свой NUTbook. Во время процесса конфигурирования, поскольку пользователь может иметь большинство своих устройств, представленных посредством записей в каталожной карте, он может иметь возможность регистрировать некоторые или все устройства, к которым он может хотеть предоставлять доступ на маршрутизаторе посредством Nut-идентификаторов. Инициирующие Nut-сообщения могут содержать Nut-идентификатор отправляющего устройства и в силу этого могут надлежащим образом сверяться с регистрационным списком для доступа. Маршрутизатор затем может инструктироваться таким образом, чтобы устанавливать отношения между различными устройствами и собой, в силу чего он может обеспечивать возможность защищенной связи для контента Nut-сообщений. Блок-схема последовательности операций способа для обработки сообщений в NUTS Wi-Fi-маршрутизаторе показана на фиг. 168. Некоторые или все сообщения, которые могут проходить через маршрутизатор посредством зарегистрированных устройств, могут аутентифицироваться. Этап 16818 показывает интересный признак, который может быть доступным на маршрутизаторах на основе NUTS. Незарегистрированное устройство может контактировать с маршрутизатором для доступа не с использованием RBK. Когда это происходит, он может искать указываемые владельцем конфигурационные настройки для выделений полосы пропускания и ограничений для различных категорий Wi-Fi-доступа: зарегистрирован, IoT и/или гость. Зарегистрированные устройства могут задаваться без ограничений на тип использования и полосы пропускания, которую запрашивают. IoT/IoN-устройства может иметь собственную категорию и может требовать идентичного уровня аутентификации в качестве зарегистрированных устройств, но может быть отдельно управляемым в качестве группы. Таблица на фиг. 169 показывает заданные категории и тип доступа, который они могут иметь через маршрутизатор. Гостевым устройствам может предоставляться доступ с использованием нормальных протоколов, но с ограничениями. Примерная конфигурация для ограничений по атрибутам на основе категории показана на фиг. 170. Владелец может указывать ограничения в расчете на устройство, такие как, но не только, истечение срока действия, полоса пропускания, совокупная полоса пропускания, максимальное число соединений типа категории, назначения и/или режимы передачи сообщений. Таким образом, гостевые устройства могут иметь доступ в Интернет через неизвестный NUTS Wi-Fi-маршрутизатор в определенных пределах, в то время как аутентифицированная NUTS-сеть intranet может защищаться посредством защищенных способов NUTS-уровня. Эта технология эффективно может создавать отдельно управляемые категории каналов в рамках инфраструктуры Wi-Fi-связи.

Некоторые или все зарегистрированные устройства пользователя теперь могут быть независимыми от внутренне назначенных IP-адресов для идентификации, а вместо этого посредством Nut-идентификаторов в каталожной карте. Это может представлять собой свойство NUTS, чтобы повышать материальность и функциональность данных и аппаратных средств в некоторых или всех сетях более универсальным способом. Маршрутизатор может отслеживать динамические назначения IP-адресов, увязанные с Nut-идентификаторами зарегистрированных устройств. В будущих итерациях и других вариантах осуществления, изготовители аппаратных средств могут обеспечивать возможность использования Nut-идентификаторов вместе с IP-адресами и/или MAC-адресами, чтобы осуществлять доступ к Ethernet-интерфейсам на различных устройствах. Nut-идентификаторы для идентификации устройств могут рассматриваться в качестве эквивалента назначения системного имени для установки ОС на PC, но они могут быть систематическими и практически уникальными, в силу чего изменение или добавление Ethernet-карты в систему позволяет представлять новые IP-адреса и/или MAC-адреса, но они не могут изменять Nut-идентификатор, ассоциированный с устройством.

Родительский контроль доступ в Интернет их детей может отслеживаться и ограничиваться на уровне маршрутизатора с использованием Wi-Fi-маршрутизатора на основе NUTS, а не или в дополнение уровням устройства и пользователя. Nut-контейнер сообщений, который может огибать трафик зарегистрированного устройства, может включать в себя пользовательскую идентификационную информацию, которая может использоваться для того, чтобы дополнительно фильтровать трафик посредством родительских предпочтений.

Обертывание приложений с Nut-контейнерами

Появление и разработка облачных услуг, магазинов приложений и/или их ассоциированных приложений может обеспечивать некоторую форму модуляризации и/или переносимости приложений на разнообразных устройствах. Тем не менее, это может не иметь место с настольными или переносными компьютерами. Большинство приложений, которые могут выполняться на них, могут требовать установок и/или обслуживания вручную. Это также может быть истинным для оптимально поддерживаемых институциональных окружений, в которых смешение заранее выбранных комплектов приложений может быть свернуто в настраиваемый установочный пакет системными администраторами для простоты настроек

машины. Альтернативно, они могут создавать клонированные предварительно установленные приложения на дисках, которые могут вставляться/выниматься в/из компьютеров. Для рабочего окружения, для людей и/или администраторов очень сложно и затруднительно отслеживать и авторизовать каждую программу, которая может устанавливаться на конкретном устройстве. Очень строгие правила формирования учетной записи могут приводить к сниженной производительности для пользователя или к растущим требованиям по персоналу для отдела разработки систем.

Приложение, обернутое в тщательно сконструированный Nut-контейнер, может разрешать множество этих сложностей. Локальные операционные системы могут модифицироваться, чтобы обеспечивать возможность только Nut-обернутым приложениям выполняться. Последствий может быть множество. Это может предотвращать некоторые или все неавторизованные установки и выполнение неутвержденных и непроверенных приложений. Политики могут принудительно активироваться посредством централизованного администрирования ключей доступа в управляемой институциональной среде. Векторы вирусного заражения, которые могут предусматривать выполнение явного двоичного файла, могут радикально уменьшаться. NUTserver-признаки репликации и синхронизации могут обеспечивать возможность легкого распространения более новых версий установленного программного обеспечения на некоторые или все устройства. Надлежащим образом обернутым Nut-контейнерам можно удаленно скомандовать на предмет того, чтобы самоустанавливаться с использованием удаленного управляющего интерфейса при успешной синхронизации. Резервные копии окружения устройства и дублирование могут быть автоматизированы с использованием NUTserver, как проиллюстрировано на фиг. 171. Вычислительное устройство 17100 может сохранять резервное копирование Nut-контейнеров для устройства, которое может сбоить. После подготовки нового устройства 17140 к установке, приложение, которое, возможно, должно устанавливаться надлежащим образом, может представлять собой NUTserver 17144 и его ключи доступа. Затем команда дублирования из вычислительных устройств с корректными ключами может инициировать копирование некоторых или из всех релевантных Nut-контейнеров из устройства 1 в устройство 2 и затем может выполнять необходимые установки некоторых или всех Nut-обернутых приложений.

Поверхностно, этот способ может казаться несильно отличающимся от клонирования накопителей на жестких дисках или наличия тщательно продуманного сценария установки, но могут быть некоторые существенные различия. Nut-обернутое приложение может быть спецификацией приложения, а не самого конкретного двоичного файла. Двоичный файл может сохраняться в институциональном MIOR, и затем механизмы MIO могут вступать во владение во время вводного процесса спецификации Nut-обернутых приложений, чтобы осуществлять выборку правильной версии приложения для текущей операционной системы устройства, которое может или может не быть идентичным исходному устройству, которое оно может заменять. Это использование MIOR может быть способом для того, чтобы управлять версиями приложений в вычислительном окружении, содержащем гетерогенные операционные системы и/или аппаратные средства.

Использование NUTS-технологии фактически может обеспечивать возможность некоторым или всем из этих процессов возникать из любого места в Интернете, в силу чего новые машины могут устанавливаться и обслуживаться от имени учреждения удаленным способом.

Пример означенного может представлять собой то, что у продавца в недельной поездке украли его переносной компьютер, который содержит 20 настраиваемых представлений и конфиденциальных клиентских отчетов, которые он хочет использовать на встречах с клиентами. При условии, что компания использует NUTS, продавец может идти в ближайший компьютерный магазин и купить заменяющий переносной компьютер под управлением системного администратора. Он затем может устанавливать стандартный NUTserver, загружаемый из Интернета, на этом переносном компьютере. Администратор может отправлять ему специально кодированный Nut-контейнер доступа/установки, называемый образующим Nut-контейнером, по электронной почте, и продавец может загружать этот образующий Nut-контейнер на свой новый переносной компьютер из корпоративной почтовой страницы на основе веб-браузера. Администратор может звонить ему и сообщать продавцу секретный фразовый пароль, который может отмыкать образующий Nut-контейнер. После размыкания с использованием локального NUTserver/NUTbrowser, образующий Nut-контейнер может инициировать некоторые или все процессы, необходимые в Интернете, чтобы дублировать приложения и данные из потерянного переносного компьютера продавца из его последних синхронизации с корпоративными серверами. В пределах от нескольких минут до нескольких часов в зависимости от объема данных в резервных копиях, продавец может быть полностью функциональным с некоторыми или всеми его контактами, приложениями и/или Nut-контейнерами данных, переустановленными на его новом переносном компьютере, и это может осуществляться на различных брендах переносных компьютеров и в различных операционных системах при условии, что корпоративный MIOR может надлежащим образом отбираться и поддерживаться. Параллельно этим работам по дублированию, администратор может отправлять команды самоудаления в украденный ноутбук для некоторых или всех Nut-контейнеров в собственности компании, сохраненных на нем, в случае если вор запускает переносной компьютер с соединением с Интернетом. Это может представлять собой меру предосторожности, поскольку Nut-контейнеры на переносном компьютере могут

уже индивидуально защищаться с помощью корпоративных политик истечения срока действия Nut-контейнера.

В другом варианте осуществления встроенный NUTserver аппаратных средств может интегрироваться в неинициализированное вычислительное устройство, которое может иметь соединение с сетью, питающей доступные исходные NUTserver и MIOR-серверы. Образующий Nut-контейнер может загружаться на устройство, и к нему может осуществляться доступ, который может инициировать процессы, которые могут приводить к полной установке на это неинициализированное вычислительное устройство вычислительного окружения, включающего в себя ОС, драйверы, приложения, конфигурационные данные приложений и/или пользовательские данные. Выбор ОС можно выходить до пользователя после анализа устройства и контента доступных MIOR-кэшей. Приложения могут устанавливаться инкрементно по мере того, как пользователь осуществляет доступ к различным Nut-контейнерам, или все сразу посредством выполнения запроса относительно исходного NUTserver на предмет полного списка необходимых приложений для осуществления доступа к Nut-контейнерам пользователя.

Услуга обработки событий (EPS) NUThub может обеспечивать возможность связи на основе Nut-контейнеров с IoN/IoT-устройствами и NUTserver. Услуга обработки событий (EPS) может функционировать в качестве координатора для архивации событий, которые могут формироваться посредством IoN-устройств и приложений, которые могут хотеть формировать событие или реагировать на него, как проиллюстрировано на фиг. 172. Поскольку некоторые или все события могут содержаться в Nut-контейнерах, любое событие может передаваться через любую сеть при условии, что между устройствами может быть обходимый маршрут. Это может обеспечивать возможность пользователю отслеживать на предмет требуемых событий в локальных и удаленных IoN/IoT-устройствах и/или NUTserver-системах. Это может обеспечивать возможность пользователю инициировать диспетчеризуемые или произвольно организующиеся события на локальных и/или удаленных устройствах. События могут реплицироваться на некоторых или всех устройствах пользователя при желании. EPS может работать с удаленным управляющим интерфейсом, чтобы предоставлять возможность инициирования конкретных для устройства команд на основе событий. Фиг. 172 осуществляет сценарий, в котором локальное приложение 17208 для работы с календарем на устройстве 17200 может инициировать синхронизированное событие через локальную EPS 17204, которая должна выполняться на IoN-устройстве 17220, которое может быть достижимым посредством NUTserver 17212 на устройстве 17210. Локальная EPS 17204 может ретранслировать событие в другую EPS 17214, которая может иметь доступ к целевому IoN-устройству 17220. EPS 17214 затем может ретранслировать событие/команду в свой локальный NUTserver 17212, и после этого она может использовать свой IoN/IoT-интерфейс для того, чтобы передавать событие/Nut-контейнер команд в IoN-устройство 17220. При приеме события/Nut-контейнера команд, IoN-устройство 17220 может аутентифицировать и затем может выполнять команду через свой удаленный управляющий интерфейс. Примеры таких событий могут варьироваться между, но не только, запуском удаленных серверов по расписанию, отправкой почтовых сообщений по расписанию, отправкой сообщений чата относительно состояний системы, завариванием кофе утром в IoN-совместимой кофеварке, изменением температурных настроек на интеллектуальном термостате и/или прогревом автомобилям холодным зимним утром через двадцать минут после того, как заканчивается заваривание кофе.

EPS может сохранять прошедшие события, которые она может принимать и формировать на каждом устройстве, на котором она может выполняться в области Nut-хранилища события 17216 и 17206. Она может выступать в качестве репозитория событий, а также очереди событий для сбоев связи и устройств. Пользователь или администратор может просматривать эти события позднее и может анализировать их для любого использования после этого. Пользователь с учетной NUTcloud-записью также может иметь свои события, реплицированные ему таким образом, что события могут просматриваться из любого доступа в Интернет. Некоторые или все события могут быть защищенным Nut-контейнером и могут принадлежать пользователю. NUThub может взаимодействовать с ними прозрачно, чтобы использовать преимущество возможностей организации очереди EPS.

Пример приложения, использующего преимущество EPS и ее репозитория, может представлять собой случай, когда система домашней аварийной сигнализации начинает предупреждение касательно того, что некоторые ее датчики с аккумуляторным питанием могут иметь низкий заряд аккумулятора. Система домашней аварийной сигнализации может формировать событие разряженного аккумулятора с указанием блока, который может быть затронут, и может запрашивать вызов по выполнению техобслуживания с компанией по аварийному техническому обслуживанию. Компания по аварийному техническому обслуживанию может предлагать различные времена, она может обслуживать проблему пользователю по электронной почте, и пользователь может выдвигать различное предположение времени или подтверждать их предлагаемое время. После подтверждения, календари в компании по аварийному техническому обслуживанию и в пользовательских устройствах могут обновляться с информацией посещения на дому автоматически. Система аварийной сигнализации может иметь ограниченную RBK-взаимосвязь с компанией по аварийному техническому обслуживанию, так что она может проводить диагностику с неявным подтверждением домовладельца защищенным способом.

Контекстные вычисления с Nut-контейнерами приложений Может возникать откровенный захват

территории для некоторых или всех граней цифровых осколков пользователя посредством веб-компаний, таких как, но не только, привычки в отношении поиска, предыстории поиска, спецификации устройств, привычки в отношении веб-просмотра, тенденции покупок, блоггинг-контент, социальные сети, бизнес-сети, почтовый контент, текстовые сообщения, фотографии и/или даже оцифрованный анализ их DNA. Подавляющее большинство этих сформированных пользователем данных не может принадлежать, быть доступными, анализироваться, изменяться, удаляться и/или управляться пользователем, который может формировать их. NUTS-технология может упрощать для разработчиков приложений сохранение сформированных пользователем данных и может упрощать выдачу копии пользователю для собственного использования и архивации. Она может предоставлять общий защищенный контейнер, который может варьироваться касательно форматов контента через MIO, чтобы предоставлять возможность индивидуальных настроек. Очень небольшое число производителей веб-услуг могут быть достаточно общими для того, чтобы охватывать большую часть аспектов цифрового отпечатка пользователя; например, Amazon может знать только некоторые из ваших предпочтений по покупкам, и Google может знать только часть вашей предыстории поиска. Таким образом, веб-производители типично могут ассемблировать частичные срезы привычек пользователя на основе услуг, которые они предоставляют. Наиболее выгодная точка, чтобы собирать некоторые или все цифровые местонахождения и действия пользователя, может представлять собой "пользователем для пользователя". На фиг. 173 показана типичная схема размещения сети для производителя и пользовательского приложения, в которой производитель может использовать локальные куки-файлы на основе браузера для того, чтобы тегировать пользователя или его текущий сеанс, и может использовать серверы сбора больших данных для того, чтобы записывать некоторые или все действия из/в приложении.

Если пользователь взаимодействует с приложениями, которые могут предоставлять полную запись своих сеансов в Nut-контейнере для собственных архивов и использования, то пользователь может в конечном счете иметь возможность собирать различные грани своих цифровых отклонений, как проиллюстрировано на фиг. 174. Эти предыстории сеансов могут предоставлять контекст, в котором анализ может осуществляться посредством контекстно-зависимых приложений, чтобы предоставлять больше удобств пользователю, как показано на фиг. 175. Приложение может сохранять свои предыстории сеансов в Nut-контейнере 17414 приложения, и они в свою очередь могут использоваться посредством некоторых или всех других приложений, которые может устанавливать пользователь, чтобы обеспечивать надлежащие преимущества пользователю. Надлежащий анализ контекста может извлекать сущность задачи, которую пользователь может хотеть выполнять. Приложение 17524 учета может записывать свои сеансы в Nut-контейнер 17414 приложения для некоторых или всех действий по оплате счетов и по чековому счету, которые может осуществлять пользователь. Приложение 17520 распознавания шаблонов, которое может читать такую предысторию сеанса, может анализировать ее и рекомендовать статистические этапы, осуществляемые для того, чтобы оплачивать ежемесячные счета, и может представлять предварительный просмотр действий, которые оно может предпринимать от имени пользователя. Если пользователь согласен с этим анализом, он может выполнять эти этапы для того, чтобы оплачивать некоторые или все релевантные счета автоматически с использованием различных учетных записей под именем пользователя. Этот Nut-контейнер приложения может быть доступен для пользователя в Интернете, если он синхронизирует свои Nut-контейнеры через NUTcloud 17500.

Другой полезный аспект контекста, сохраненного посредством Nut-контейнеров приложений, может представлять собой аспект повторяющихся процедур. Он может представлять собой общий признак для интерфейсов командной строки, которые могут предпочитать разработчики, в которых предыдущие команды могут сохраняться для необязательного повторного выполнения по запросу. Nut-контейнеры приложений могут предоставлять идентичный тип процедурных повторных вызовов по запросу для пользователя фактически в любом совместимом приложении. Приложение для поиска туров с сохранением контекста может предоставлять сущность требований для предложенного прохождения в Nut-контейнере приложения после начального поиска в веб пользователем. Позднее, пользователь может возобновлять этот поиск в некоторых или всех своих предпочтительных туристических фирмах автоматически посредством повторного выполнения дистиллированных требований по ним с использованием приложения для контекстно-зависимого поиска туров. Это может уменьшать время, потраченное на повторный ввод в различные формы на каждом веб-узле поиска туров, и может формировать автоматическую сводку некоторых или всех его опций. Кроме того, поскольку процесс может полностью управляться пользователем, и часть или вся конфиденциальная информация могут сохраняться посредством его NUTbook, запросы к производителям, для которых он может иметь привилегии за накопленные мили и/или членства, могут применяться надлежащим образом посредством приложения для контекстно-зависимого поиска туров, чтобы получать самые персонифицированные и значимые результаты для него. Этот тип глубокого контекстно-зависимого поиска может быть фактически невозможно осуществлять одним производителем, если пользователь искренне не может предоставлять беспрепятственный доступ к части или ко всей своей чувствительной цифровой информации в некоторых или всех случаях для этого производителя и полностью доверять ему; это может быть очень сомнительным предложением для среднестатистического пользователя с цифровым восприятием.

В другом варианте осуществления фиг. 176 показывает схему размещения сети для IoN/IoT-устройств пользователя и различных коммунальных и других услуг, на которые он может подписываться в своей повседневной жизни дома. Ни одна компания не может иметь возможность собирать всю домашнюю жизнь пользователя цифровым способом. Тем не менее, пользователь может достигать этого, если некоторые или все его устройства формируют Nut-контейнеры приложений, и он имеет приложение, которое может анализировать его различные цифровые контексты.

Энергосберегающее контекстно-зависимое приложение может анализировать использование электричества посредством различных электронных приборов в его доме и может объединять его дневным и ночным тарифами электроэнергетической компании, чтобы предлагать показатели энергосбережения, которые могут быть автоматически осуществлены посредством приложения от его имени. Оно может анализировать его привычки при персональном использовании каждого устройства, чтобы координировать удобные комбинации для него, когда это распознает стечение обстоятельств из прошлого. IoN/IoT-устройства могут сообщать ему требования по обслуживанию, если периодически выполняемая самодиагностика раскрывает сбойные части или субоптимальные оперативные показания.

Могут возникать проблемы безопасности с IoT-устройствами, содержащими различные датчики состояния окружающей среды, которые могут не управляться полностью владельцем устройства, а вместо этого изготовителями и/или потенциальными злоумышленными хакерами. Фиг. 177 показывает пример схемы размещения сети двух IoN-устройств и их соответствующих изготовителей. Когда Nut-контейнеры приложений 17734 и 17744 могут формироваться посредством каждого IoN-устройства 17730 и 17740, оно может локально архивироваться посредством NUTserver 17722 в локальном хранилище 17724. Эти архивированные Nut-контейнеры приложений в дальнейшем могут анализироваться и фильтроваться пользователем перед пересылкой их изготовителям, чтобы удалять любую конфиденциальную информацию, которую пользователь считает несоответствующей для сбора посредством третьей стороны. На фиг. 178 контекстное аналитическое приложение 17808 может предлагать специализированную регламентную фильтрацию некоторых или всех своих IoN/IoT-сформированных сообщений, чтобы минимизировать ненамеренное раскрытие своей конфиденциальности для третьих сторон. Таким образом, третьи стороны по-прежнему могут собирать некоторые данные из каждого проданного устройства только в той степени, в которой может разрешать каждый владелец; в силу этого они могут логически выводить то, какую персональную информацию может быть готов предоставлять им среднестатистический покупатель.

Заключение и философия

Различные варианты осуществления и примеры сценариев, которые детализированы, могут быть основаны на такой базовой NUTS-философии, что данные принадлежат пользователю, который их формирует, и что пользователь может иметь средства для того, чтобы точно управлять их раскрытием. Проектное решение может быть достаточно гибким, чтобы приспосабливать варьирования и/или альтернативы, такие как, но не только, альтернативные способы шифрования, ключи различных длин, различные превращения данных и/или различные замыкающие механизмы. SDFT предоставляет полезный набор инструментальных средств для программиста, с помощью которого можно превращать данные на самых нижних уровнях и можно помогать в обеспечении возможности структурированного криптографического программирования конструировать NUTS-структуры и другие сложные криптографические структуры. SDFT обеспечивает портативность данных, спаренных с его командами превращения, гибким и обобщенным способом. Различные варианты осуществления NUT-контейнеров могут индивидуально настраиваться, чтобы вписываться в существующие организационные инфраструктуры и инфраструктуры безопасности, или они могут представлять собой автономные установки для отдельного пользователя. Материальность данных может представлять собой важную философию, которую предлагает NUTS, и может реализовывать способность пользователей сохранять, манипулировать и/или анализировать данные, которые они могут формировать простыми способами при предложении признаков, приличествующих самым сложным управляемым системам. В заключение, NUTS может предоставлять отдельным пользователям альтернативу современным способам организации их цифровых работ и данных.

ФОРМУЛА ИЗОБРЕТЕНИЯ

1. Компьютерно-реализуемый способ обработки данных, содержащий этапы, на которых

осуществляют посредством по меньшей мере одного процессора доступ к структуре хранения данных в памяти, причем структура хранения данных содержит по меньшей мере одно поле данных, включающее в себя по меньшей мере один входной объект данных, и по меньшей мере одно поле команд, включающее в себя множество команд преобразования, которые должны выполняться в последовательности, заданной в структуре хранения данных, в отношении одного и того же из упомянутого по меньшей мере одного входного объекта данных; и

выполняют упомянутую заданную последовательность посредством выполнения каждой из упомянутого множества команд преобразования в прямом порядке этой последовательности в отношении упомянутого по меньшей мере одного входного объекта данных с возможностью формирования по меньшей мере одного выходного объекта данных, причем каждое преобразование выполняется в прямом режиме в отношении данных и атрибутов с получением преобразованных данных и атрибутов; и

заменяют упомянутый по меньшей мере один входной объект данных в упомянутом по меньшей мере одном поле данных структуры хранения данных упомянутым по меньшей мере одним выходным объектом данных, оставляя при этом упомянутое множество команд преобразования в упомянутом по меньшей мере одном поле команд структуры хранения данных, причем данное множество команд преобразования составлены с возможностью применения к упомянутому по меньшей мере одному выходному объекту данных для восстановления упомянутого по меньшей мере одного входного объекта данных.

2. Способ по п.1, дополнительно содержащий этапы, на которых

выполняют упомянутую заданную последовательность посредством выполнения каждой из упомянутого множества команд преобразования в обратном порядке этой последовательности в отношении упомянутого, по меньшей мере, выходного объекта данных для формирования упомянутого по меньшей мере одного входного объекта данных, причем каждое преобразование выполняется в обратном режиме в отношении преобразованных данных и атрибутов для получения данных и атрибутов; и

заменяют упомянутый по меньшей мере один выходной объект данных в структуре хранения данных упомянутым по меньшей мере одним входным объектом данных, оставляя при этом упомянутое множество команд преобразования в структуре хранения данных.

3. Способ по п.2, в котором по меньшей мере одна из упомянутого множества команд преобразования, выполняемых в обратном режиме, содержит обработку упомянутого по меньшей мере одного входного объекта данных в прямом режиме, для формирования второго по меньшей мере одного выходного объекта данных и сравнение упомянутого по меньшей мере одного выходного объекта данных со вторым по меньшей мере одним выходным объектом данных для получения результата верификации.

4. Способ по п.1, в котором по меньшей мере одна из упомянутого множества команд преобразования, выполняемых в прямом режиме, требует по меньшей мере один атрибут для обработки упомянутого по меньшей мере одного входного объекта данных для формирования по меньшей мере одного выходного объекта данных.

5. Способ по п.4, в котором по меньшей мере одна из упомянутого множества команд преобразования, выполняемых в обратном режиме, требует по меньшей мере один атрибут для обработки упомянутого по меньшей мере одного выходного объекта для формирования упомянутого по меньшей мере одного входного объекта данных.

6. Способ по п.4, в котором по меньшей мере один процессор генерирует по меньшей мере один атрибут, требующийся для по меньшей мере одной из упомянутого множества команд преобразования, выполняемых в прямом режиме для обработки упомянутого по меньшей мере одного входного объекта данных для формирования по меньшей мере одного выходного объекта данных.

7. Способ по п.5, в котором по меньшей мере одна из упомянутого множества команд преобразования, выполняемых в обратном режиме, требует по меньшей мере один атрибут для обработки упомянутого по меньшей мере одного входного объекта данных для формирования по меньшей мере одного выходного объекта данных, содержащего результат верификации.

8. Способ по п.4, дополнительно содержащий этап, на котором проверяют посредством по меньшей мере одного процессора структуру по меньшей мере одного атрибута, представленного для использования по меньшей мере одной из упомянутого множества команд преобразования, выполняемых в прямом режиме, которая требует по меньшей мере один атрибут конкретной структуры для обработки упомянутого по меньшей мере одного входного объекта данных для формирования упомянутого по меньшей мере одного выходного объекта данных.

9. Способ по п.8, дополнительно содержащий этап, на котором проверяют посредством по меньшей мере одного процессора структуру по меньшей мере одного атрибута, представленного для использования по меньшей мере одной из упомянутого множества команд преобразования, выполняемых в обратном режиме, которая требует по меньшей мере один атрибут конкретной структуры для обработки упомянутого по меньшей мере одного выходного объекта данных для формирования упомянутого по мень-

шей мере одного входного объекта данных.

10. Способ по п.1, в котором по меньшей мере одна из упомянутого множества команд преобразования преобразует упомянутую структуру хранения данных во вторую структуру хранения данных, имеющую структуру, отличающуюся от структуры упомянутой структуры хранения данных.

11. Способ по п.10, в котором по меньшей мере одна из упомянутого множества команд преобразования представляет собой команду преобразование Мебиуса.

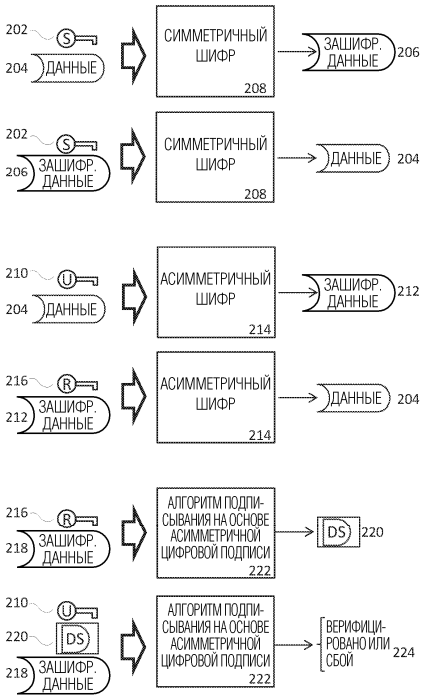
12. Способ по п.1, в котором упомянутая структура хранения данных предоставляется в качестве по меньшей мере одного входного объекта данных во второй структуре хранения данных; и по меньшей мере один процессор осуществляет доступ ко второй структуре хранения данных для обработки второй структуры хранения данных.

13. Способ по п.1, в котором две или более последовательных команд преобразования из упомянутого множества команд преобразования формируют зависимую группу; и каждая команда преобразования в зависимой группе обрабатывается в одинаковом прямом порядке.

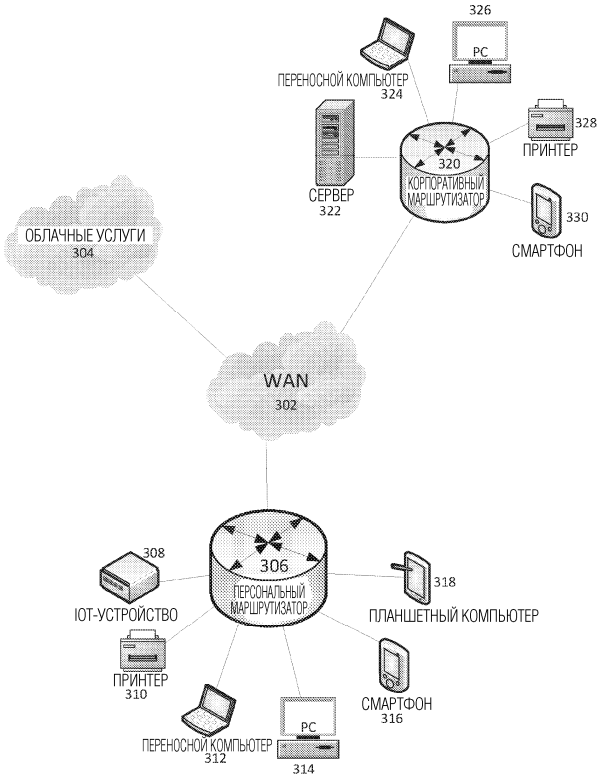
14. Способ по п.3, в котором по меньшей мере одна из упомянутого множества команд преобразования, выполняемых в обратном режиме, выдает сбой верификации и завершает обработку этого множества команд преобразования.

ТИП КЛЮЧА	СИМВОЛ
ФРАЗОВЫЙ ПАРОЛЬ	P 102
СИММЕТРИЧНЫЙ	S 104
АСИММЕТРИЧНЫЙ	A 106
АСИММЕТРИЧНЫЙ ОТКРЫТЫЙ	U 108
АСИММЕТРИЧНЫЙ ЗАКРЫТЫЙ	R 110

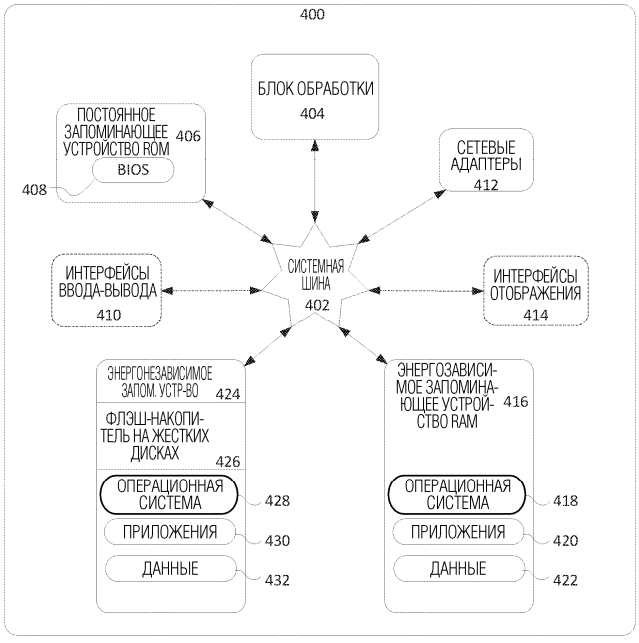
Фиг. 1



Фиг. 2



Фиг. 3



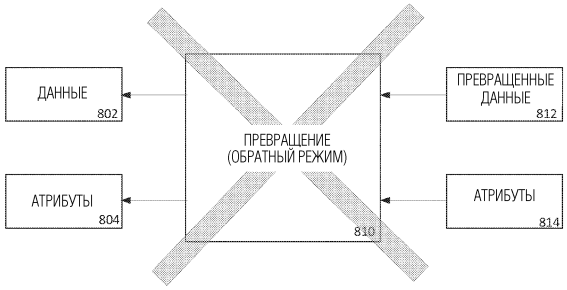
Фиг. 4

Фиг. 5

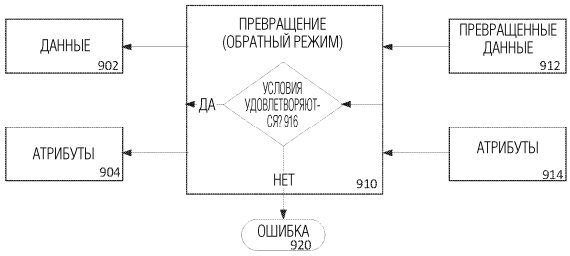
	ПРЕВРАЩЕНИЕ		
ОПЕРАЦИЯ С ДАННЫМИ	**КЛАССИФИКАЦИЯ**	**ОПЕРАЦИЯ**	**ВЫПОЛНЯЕМАЯ КОМАНДА**
JSON-ПРЕОБРАЗОВАНИЕ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ	ПРЕОБРАЗОВАНИЕ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ	json	serialize json
СЖАТИЕ ДАННЫХ	СЖАТИЕ	Zlib	compress zlib
BASE64-КОДИРОВАНИЕ	КОДИРОВАНИЕ	base 64	encode base 64
КОДИРОВАНИЕ В ДВОИЧНОЙ ФОРМЕ В UTF8	КОДИРОВАНИЕ	strbin utf_8	encode strbin utf_8
АСИММЕТРИЧНЫЙ ШИФР	А-ШИФР	pkcsl_oaep	acipher pkcsl_oaep
СИММЕТРИЧНЫЙ ШИФР	S-ШИФР	aes	scipher aes
ФУНКЦИЯ ИЗВЛЕЧЕНИЯ КЛЮЧА	ИЗВЛЕЧЕНИЕ	pbkdf2	derive pbkdf2
ФУНКЦИЯ РАСТЯГИВАНИЯ КЛЮЧА	ИЗВЛЕЧЕНИЕ	hkdf	derive hkdf
ХЭШ	ДАЙДЖЕСТ	hash	digest hash
MAC	ДАЙДЖЕСТ	hash	digest hash
HMAC	ДАЙДЖЕСТ	hmac	digest hmac
CMAC	ДАЙДЖЕСТ	cmac	digest cmac
ЦИФРОВАЯ PSS-ПОДПИСЬ	Ц-ПОДПИСЬ	pkcsl_pss	dign pcksl_pss

Фиг. 6

Фиг. 7



Фиг. 8



Фиг. 9

ПРЕВРАЩЕНИЕ	ОПЕРАЦИИ
ПРЕОБРАЗОВАНИЕ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ	JSON, XML, COM, CORBA, SOAP
СЖАТИЕ	Zlib, gzip, bz2, izma
КОДИРОВАНИЕ	base64, base85, utf_8, quopri, двоичные, более чем 90 вариантов кодеков
A-ШИФР	pkcs1_oaep, pkcs1_v1_5
S-ШИФР	aes, chacha20, saisa20
ИЗВЛЕЧЕНИЕ	pbkdf2, hkdf, scrypt
ДАЙДЖЕСТ	hash: crc, md5, sha1, sha2, shae3, shake128, shake256, keccak hmac: md5, sha1, sha2 cmac: aes
Ц-ПОДПИСЬ	pkcsl_v1_5, pkcs1_pss, dss

Фиг. 10

Типы кодеков Python v. 3.6

ascii	cp863	cp1256	iso8859_2	mac_greek
big5	cp864	cp1257	iso8859_3	mac_iceland
big5hkscs	cp865	cp1258	iso8859_4	mac_latin2
cp037	cp866	cp65001	iso8859_5	mac_roman
cp273	cp869	euc_jp	iso8859_6	mac_turkish
cp424	cp874	euc_jis_2004	iso8859_7	ptcp154
cp437	cp875	euc_jisx0213	iso8859_8	shift_jis
cp500	cp932	euc_kr	iso8859_9	shift_jis_2004
cp720	cp949	gb2312	iso8859_10	shift_jisx0213
cp737	cp950	gbk	iso8859_11	utf_32
cp775	cp1006	gb18030	iso8859_13	utf_32_be
cp850	cp1026	hz	iso8859_14	utf_32_le
cp852	cp1125	iso2022_jp	iso8859_15	utf_16
cp855	cp1140	iso2022_jp_1	iso8859_16	utf_16_be
cp856	cp1250	iso2022_jp_2	johab	utf_16_le
cp857	cp1251	iso2022_jp_2004	koi8_r	utf_7
cp858	cp1252	iso2022_jp_3	koi8_t	utf_8
cp860	cp1253	iso2022_jp_ext	koi8_u	utf_8_sig
cp861	cp1254	iso2022_kr	kz1048	
cp862	cp1255	latin_1	mac_cyrillic	

Фиг. 11

ПРЕВРАЩЕНИЕ	ОПИСАНИЕ
ПРЕОБРАЗОВАНИЕ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ	ПРЕОБРАЗОВАНИЕ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ СТРУКТУРЫ ДАННЫХ
СЖАТИЕ	СЖАТИЕ ДАННЫХ
КОДИРОВАНИЕ	КОДИРОВАНИЕ СТРОКИ/БАЙТОВ СОГЛАСНО СПЕЦИФИКАЦИИ
A-ШИФР	ШИФРОВАНИЕ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ АСИММЕТРИЧНОГО СПОСОБА
S-ШИФР	ШИФРОВАНИЕ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ СИММЕТРИЧНОГО СПОСОБА
ИЗВЛЕЧЕНИЕ	ФУНКЦИЯ ИЗВЛЕЧЕНИЯ КЛЮЧА
ДАЙДЖЕСТ	СОЗДАНИЕ ДАЙДЖЕСТА ДЛЯ ДАННЫХ
Ц-ПОДПИСЬ	ЦИФРОВАЯ ПОДПИСЬ
КЛЮЧ	МАНИПУЛИРОВАНИЯ И УПРАВЛЕНИЕ КЛЮЧАМИ
ОЧИСТКА	ОБСЛУЖИВАНИЕ ВНУТРЕННИХ СТРУКТУР
TAR-ГРУППА	ГРУППИРОВКИ В TAR ДЛЯ ЗАВИСИМЫХ ПРЕВРАЩЕНИЙ
ПРИЖАТИЕ	КОНКРЕТНАЯ ДЛЯ ЯЗЫКА ПОДГОТОВКА ДАННЫХ
ЗАМОК	ЗАМЫКАНИЕ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ИЗМЕНЯЕМОГО ЗАМКА
МЕБИУС	ПРЕОБРАЗОВАНИЕ ИЗ ОДНОЙ СТРУКТУРЫ В ДРУГУЮ ВНУТРЕННЕ

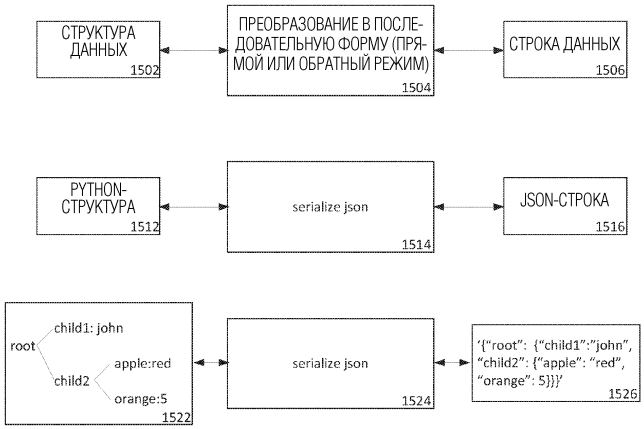
Фиг. 12

	ОБРАТИМОСТЬ		
ПРЕВРАЩЕНИЕ	**ОБРАТИМОЕ**	**НЕОБРАТИМОЕ**	**УСЛОВНОЕ**
ПРЕОБРАЗОВАНИЕ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ	✓		
СЖАТИЕ	✓	✓	
КОДИРОВАНИЕ	✓		
A-ШИФР			✓
S-ШИФР			✓
ИЗВЛЕЧЕНИЕ		✓	
ДАЙДЖЕСТ		✓	
Ц-ПОДПИСЬ		✓	
КЛЮЧ		✓	
ОЧИСТКА		✓	
TAR-ГРУППА		✓	
ПРИЖАТИЕ	✓		
ЗАМОК			✓
МЕБИУС	✓		

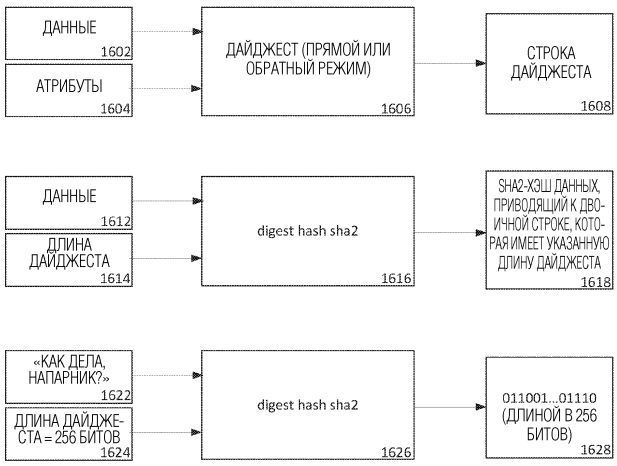
Фиг. 13

	РЕЖИМ	
ПРЕВРАЩЕНИЕ	**ПРЯМОЕ**	**ОБРАТНОЕ**
ПРЕОБРАЗОВАНИЕ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ	ПРЕОБРАЗОВАНИЕ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ	ОТМЕНА ПРЕОБРАЗОВАНИЯ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ
СЖАТИЕ	СЖАТИЕ	РАСПАКОВКА
КОДИРОВАНИЕ	КОДИРОВАНИЕ	ДЕКОДИРОВАНИЕ
A-ШИФР	ШИФРОВАНИЕ	ДЕШИФРОВАНИЕ
S-ШИФР	ШИФРОВАНИЕ	ДЕШИФРОВАНИЕ
ИЗВЛЕЧЕНИЕ	ИЗВЛЕЧЕНИЕ	ИЗВЛЕЧЕНИЕ
ДАЙДЖЕСТ	ДАЙДЖЕСТ	ВЕРИФИКАЦИЯ
Ц-ПОДПИСЬ	ПОДПИСЫВАНИЕ	АУТЕНТИФИКАЦИЯ
КЛЮЧ	КЛЮЧ	
ОЧИСТКА	ОЧИСТКА	
TAR-ГРУППА	TAR-ГРУППА	TAR-ГРУППА
ПРИЖАТИЕ	ПРИЖАТИЕ	ОТЖАТИЕ
ЗАМОК	ЗАМЫКАНИЕ	ОТМЫКАНИЕ
МЕБИУС	МЕБИУС	МЕБИУС

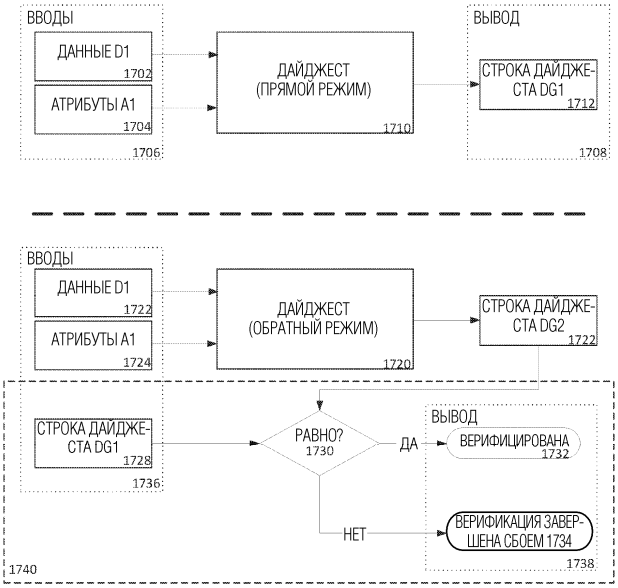
Фиг. 14



Фиг. 15

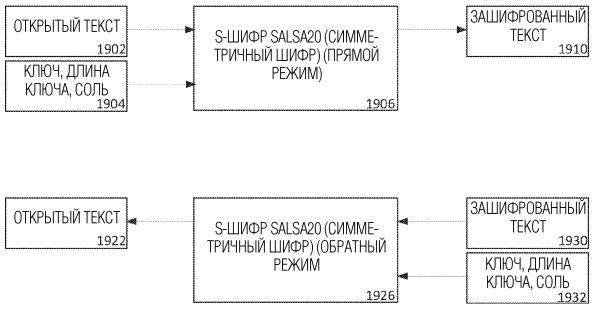


Фиг. 16

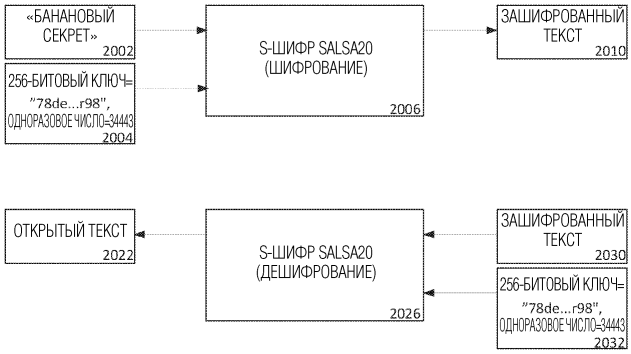


Фиг. 17

Фиг. 18



Фиг. 19



Фиг. 20

2102

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	КЛЮЧИ СОРТИРОВКИ=	ВВОД	ВЫВОД
ПРЕОБРАЗОВАНИЕ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ	json	[tif]	СТРУКТУРА	СТРОКА
ПРЕОБРАЗОВАНИЕ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ	xml	[tif]	СТРУКТУРА	СТРОКА
СЖАТИЕ	zlib (gzip)	-	ДВОИЧНЫЙ	ДВОИЧНЫЙ
СЖАТИЕ	bz2	-	ДВОИЧНЫЙ	ДВОИЧНЫЙ
СЖАТИЕ	izma	-	ДВОИЧНЫЙ	ДВОИЧНЫЙ

2104
КОМАНДЫ ПРЕВРАЩЕНИЯ

```
serialize json
serialize json t
serialize json sortkeys=t
serialize operation=json sortkeys=t
compress bz2
compress operation=lzma
```

Фиг. 21

2202

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	КОДИРОВАНИЕ=	ВВОД	ВЫВОД
КОДИРОВАНИЕ	strbin	КОДЕКИ (98), ПО УМОЛЧАНИЮ utf_8	СТРОКА	ДВОИЧНЫЙ
КОДИРОВАНИЕ	utf	КОДЕКИ (98), ПО УМОЛЧАНИЮ utf_8	СТРОКА	СТРОКА
КОДИРОВАНИЕ	binascii	[hexl64lqpluu]	ДВОИЧНЫЙ	СТРОКА
КОДИРОВАНИЕ	ОСНОВАНИЕ	[16l32l64urll64stdl85la85]	ДВОИЧНЫЙ	СТРОКА
КОДИРОВАНИЕ	quopri	КОДЕКИ (98), ПО УМОЛЧАНИЮ utf_8	ДВОИЧНЫЙ	СТРОКА
КОДИРОВАНИЕ	codecs	ЛЮБОЕ КОДИРОВАНИЕ, К ПРИМЕРУ, ROT_13 ИЗ PYTHON-МОДУЛЯ ОНО ИМЕЕТ 98+10 ДОПОЛНИТЕЛЬНЫХ КОДИРОВАНИЙ	СТРОКА СТРОКА ДВОИЧНЫЙ	СТРОКА ДВОИЧНЫЙ ДВОИЧНЫЙ

2204
КОМАНДЫ ПРЕВРАЩЕНИЯ

```
encode strbin utf_8
encode utf utf_16
encode binascii hex
encode base 64
encode quopri utf_8
encode utf_8
```

Фиг. 22

2302

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	ТИП.=	ДЛ. ДАЙДЖЕСТА=	ДЛ. КЛЮЧА=	ВВОД	ВЫВОД
ДАЙДЖЕСТ	hash	crc	[16|**32**]		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hash	md5	128		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hash	md4	128		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hash	md2	128		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hash	sha1	160		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	**hash**	**sha2**	[224|256|384|**512**]		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hash	sha3	[224|256|384|**512**]		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hash	shake128	[n|**512**]		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hash	shake256	[n|**512**]		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hash	keccak	[224|256|384|**512**]		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hmac	md5	128		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hmac	sha1	160		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	hmac	sha2	[224|256|384|**512**]		СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА
ДАЙДЖЕСТ	cmac	aes	128	[128|192|**256**]	СТРОКА ДАННЫХ	СТРОКА ДАЙДЖЕСТА

2304
КОМАНДЫ ПРЕВРАЩЕНИЯ

```
digest hash md5 128
digest hash sha2 512
digest hash shake256 digestlen=332
digest hmac sha2 256
digest cmac aes 256
```

Фиг. 23

2402

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	ДЛ. КЛЮЧА=	ТИП ХЭША=	ДЛ. ДАЙДЖЕСТА	e=	ТИП КЛЮЧА
acipher	**pkcs1_oaep**	[1024|**2048**|3072]	**sha1/2/3** (9)	[160|224|256|384|**512**]	**65537**	RSA
	pkcs1_v1_5		-	-	-	

2404

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	ДЛ. КЛЮЧА=	ТИП ХЭША=	ДЛ. ДАЙДЖЕСТА	ДЛ. СОЛИ	ТИП КЛЮЧА
dign	pkcs1_v1_5	[1024|**2048**|3072]	any	[16-**512**]	-	RSA
	pkcs1_pss				**digestLen**	

2406

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	ДЛ. КЛЮЧА=	ТИП ХЭША=	ДЛ. ДАЙДЖЕСТА	РЕЖИМ=	ТИП КЛЮЧА
Ц-ПОДПИСЬ	dss	[1024|**2048**|3072]	**sha1/2** (5)	[160-**512**]	[**fips-186-3**|ДЕТЕРМИНИРОВАННЫЙ-rfc6979]	DSA
		256 (curve-256)				ECC

2410
КОМАНДЫ ПРЕВРАЩЕНИЯ

```
acipher pkcs1_oaep 2048
acipher pkcs1_oaep 1024 hashtyp=sha2
acipher pkcs1_v1_5 3072
dign pkcs1_v1_5 2048
dign pkcs1_pss 3072
dign dss 1024 hashtyp=sha2
dign dss 256
```

Фиг. 24

2502

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	ТИП ХЭША=	ДЛ. ДАЙДЖЕСТА=	ТИП СОЛИ=	ДЛ. СОЛИ=	ДЛ. КЛЮЧА=	ИТЕРАЦИИ=	ВВОД
ИЗВЛЕЧЕНИЕ	**pbkdf2**	ЛЮБОЕ	ЛЮБОЕ:**512**	И.В. СТР.	ДЛ. ДАЙДЖЕСТА=	ЛЮБОЕ:256	**10000**	СТР.

2504

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	ТИП ХЭША=	ДЛ. ДАЙДЖЕСТА=	ТИП СОЛИ=	ДЛ. СОЛИ=	ДЛ. КЛЮЧА=	ЧИС. КЛЮЧЕЙ=	ГЛАВНЫЙ
ИЗВЛЕЧЕНИЕ	hkdf	md5/**sha**1/**2**/3 (11)	[128:512]	iv	ДЛ. ДАЙДЖЕСТА=	ЛЮБОЕ:256	1+	БАЙТЫ

2506

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	ТИП СОЛИ=	ДЛ. СОЛИ=	ДЛ. КЛЮЧА=	n=	r=	p=	ЧИС. КЛЮЧЕЙ=	РЕЖИМ=	ВВОД
ИЗВЛЕЧЕНИЕ	scrypt	И.В. СТР.	ЛЮБОЕ:**512**	ЛЮБОЕ:256	16384	8	1	1+		СТР.
ИЗВЛЕЧЕНИЕ	scrypt	И.В. СТР.	ЛЮБОЕ:512	ЛЮБОЕ:256	1048576	8	1	1+	ФАЙЛ	СТР.
ИЗВЛЕЧЕНИЕ	scrypt	И.В. СТР.	ЛЮБОЕ:512	ЛЮБОЕ:256	x	y	z	1+		СТР.

ВХОД В УЧЕТНУЮ ЗАПИСЬ

ИНДИВИДУАЛЬНО НАСТРАИВАЕМЫЙ

2510

КОМАНДЫ ПРЕВРАЩЕНИЯ

```
derive pbkdf2 keylen=256 iterations=100000
derive hkdf keylen=256 numkeys=4
derive scrypt keylen=128 mode=login
derive scrypt keylen=256 mode=file
```

Фиг. 25

2602

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	ДЛ. КЛЮЧА=	РЕЖИМ=	ТИП СОЛИ=	ДЛ. СОЛИ=	ДОПОЛНЕНИЕ	Block	ТИП
S-ШИФР	aes	[128|192|**256**]	ecb	-	-	ДА	128	БЛОК
S-ШИФР	aes	[128|192|**256**]	cbc	И.В.	128	ДА	128	БЛОК
S-ШИФР	aes	[128|192|**256**]	ctr	ОДНОРАЗОВОЕ ЧИСЛО	64	НЕТ	128	БЛОК
S-ШИФР	aes	[128|192|**256**]	cfb	И.В.	128	НЕТ	128	ПОТОК
S-ШИФР	**aes**	[128|192|**256**]	**ofb**	И.В.	128	НЕТ	128	ПОТОК
S-ШИФР	aes	[128|192|**256**]	ocb	ОДНОРАЗОВОЕ ЧИСЛО	120	НЕТ	128	AEAD
S-ШИФР	aes	[128|192|**256**]	ccm	ОДНОРАЗОВОЕ ЧИСЛО	88	НЕТ	128	AEAD
S-ШИФР	aes	[128|192|**256**]	eax	ОДНОРАЗОВОЕ ЧИСЛО	128	НЕТ	128	AEAD
S-ШИФР	aes	[128|192|**256**]	gcm	ОДНОРАЗОВОЕ ЧИСЛО	128	НЕТ	128	AEAD
S-ШИФР	aes	[**256**|384|512]	siv	ОДНОРАЗОВОЕ ЧИСЛО	128	НЕТ	128	AEAD
S-ШИФР	chacha20	**256**	-	ОДНОРАЗОВОЕ ЧИСЛО	64	НЕТ	n/a	ПОТОК
S-ШИФР	salsa20	[128|**256**]	-	ОДНОРАЗОВОЕ ЧИСЛО	64	НЕТ	n/a	ПОТОК

2604

КОМАНДЫ ПРЕВРАЩЕНИЯ

```
scipher aes 256 mode=ofb
scipher aes 128 mode=gcm
scipher chacha20 256
scipher salsa20 128
```

Фиг. 26

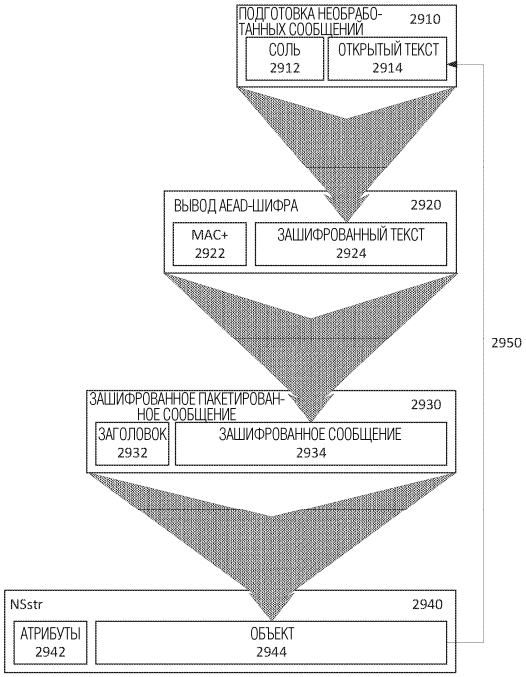
	ОПИСАНИЕ		ПРОСТОЙ ТЕКСТ	
ЭТАП 1	ЧАСТЬ		СООБЩЕНИЕ	ДОПОЛНЕНИЕ (НЕОБЯЗАТЕЛЬНОЕ)
	ДЛИНА		ПЕРЕМЕННАЯ	ДЛИНА ДОПОЛНЕНИЯ

	ОПИСАНИЕ	ПАКЕТИРОВАННОЕ СООБЩЕНИЕ			
ЭТАП 2	ЧАСТЬ	РАЗМЕР ЗАГОЛОВКА	ЗАГОЛОВОК	ЗАШИФРОВАННОЕ СООБЩЕНИЕ	
	ДЛИНА	2 БАЙТА	РАЗМЕР ЗАГОЛОВКА	ПЕРЕМЕННОЕ	
	ТИП ДАННЫХ	ДВОИЧНЫЙ	UTF8-КОДИРОВАННЫЙ	ДВОИЧНЫЙ	

Фиг. 27

КЛЮЧЕВОЕ СЛОВО	ФОРМАТ	ТИП	ЕДИНИЦЫ	ТИПИЧНЫЕ ЗНАЧЕНИЯ
РАЗМЕР ЗАГОЛОВКА	16-БИТОВЫЙ	ЦЕЛОЧИСЛЕННЫЙ	ДВОИЧНЫЙ	БЕЗ КЛЮЧЕВОГО СЛОВА В НАЧАЛЬНОМ ЗАГОЛОВКЕ, РАЗМЕР ГЛАВНОЙ СТРОКИ ЗАГОЛОВКА
key_len	НУМЕРОВАННАЯ СТРОКА	ЦЕЛОЧИСЛЕННЫЙ	БАЙТЫ	16
ciphername	СТРОКА	СТРОКА		aes/salsa20/chacha20
ciphertype	СТРОКА	СТРОКА		block/stream/aead
ciphermode	СТРОКА	СТРОКА		siv/ofb
block_len	НУМЕРОВАННАЯ СТРОКА	ЦЕЛОЧИСЛЕННЫЙ	БАЙТЫ	16
padstyle	СТРОКА	СТРОКА		char/ansix923/iso10126/pkcs7/iso7816-4/zero/space
padchar	НУМЕРОВАННАЯ СТРОКА	ЦЕЛОЧИСЛЕННЫЙ		0/32/90 -> char(0/32/90)
pad_len	НУМЕРОВАННАЯ СТРОКА	ЦЕЛОЧИСЛЕННЫЙ	БАЙТЫ	7
pad_val	ДВОИЧНАЯ СТРОКА	base64		G8934fdfs0ke
saltsource	СТРОКА	СТРОКА		cipher/stack/internal
saltfunc	СТРОКА	СТРОКА		somesaltfuncname
salttype	СТРОКА	СТРОКА		iv/nonce
salt_len	НУМЕРОВАННАЯ СТРОКА	ЦЕЛОЧИСЛЕННЫЙ	БАЙТЫ	8
salt_val	ДВОИЧНАЯ СТРОКА	base64		7efiglsi32
macsource	СТРОКА	СТРОКА		cipher/internal
macfunc	СТРОКА	СТРОКА		somemacname
mac_len	НУМЕРОВАННАЯ СТРОКА	ЦЕЛОЧИСЛЕННЫЙ	БАЙТЫ	16
mac_val	ДВОИЧНАЯ СТРОКА	base64		D34fdfs0ke

Фиг. 28



Фиг. 29

3002

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	ДЛ. КЛЮЧА ЗАМКА=	ЧИС. КЛЮ-ЧЕЙ=	ПОРОГОВОЕ ЗНАЧЕНИЕ=	S-ШИФР=	ДЛ. КЛЮЧА S-ШИФРА=	РЕЖИМ S-ШИФ-РОВАНИЯ=	ДЛ. РАБОЧИХ ДАННЫХ
ЗАМОК	sslock_b	256 (ИНДЕКС+КЛЮЧ)	n >= 2	k <= n	ЛЮБОЕ:**AES**	128	**EAX** {10}	128
ЗАМОК	sslock	256	n >= 2	k <= n	ЛЮБОЕ:**AES**	256	**EAX** {10}	256
ЗАМОК	**orlock**	[128|192|**256**]	**1**	**1**	ЛЮБОЕ:**AES**	{128-512:**256**}	**EAX** {10}	any
ЗАМОК	matlock	[128|192|256]	n >= 2	n	ЛЮБОЕ:**AES**	{128-512:**256**}	**EAX** {10}	any
ЗАМОК	xorlock	[128|192|256]	n >= 2	n	ЛЮБОЕ:**AES**	{128-512:**256**}	**EAX** {10}	any
ЗАМОК	hashlock*	[128|192|256]	n >= 2	n	ЛЮБОЕ:**AES**	{128-512:**256**}	**EAX** {10}	any

3010
КОМАНДЫ ПРЕВРАЩЕНИЯ

```
lock sslock_b 256 numkeys=6 threshold=3
lock sslock 256 numkeys=4 threshold=2
lock orlock 128 numkeys=10 scipherkeylen=128
lock matlock 256 numkeys=5
lock xorlock 128 numkeys=6
lock hashlock 192 numkeys=7
```

Фиг. 30

3102

структура	поле	тип	значения	определение
nsbin	typ	строка	"nsbin"	спецификатор типа структуры
	b64	строка	данные	данные в base64-кодировании

3104

Структура	Поле	Тип	Значения	Определение
NSjson	typ	строка	"NSjson"	спецификатор типа структуры
	obj	строка	данные	JSON-отформатированная строка

3106

Структура	Поле	Тип	Значения	Определение
NStar	typ	строка	"NStar"	спецификатор типа структуры
	name	строка		TAR-метка
	cmds	список строк		Список TAR-команд – нормальная форма
	expd	список строк		Список TAR-команд – расширенная форма

3108

Структура	Поле	Тип	Значения	Определение
NSstr	typ	строка	"NSstr"	спецификатор типа структуры
	state	строка	"preTAR" "postTAR"	Состояние структуры
	obj	значение или указатель	данные	Входные/выходные данные
	digest	двоичная строка	дайджест	Строка дайджеста obj
	keystack	Список KISS-структур		Криптографические ключи в KISS-структурах, которые должны использоваться посредством TAR
	tar	NStar-структура		Структура TAR, хранящая TAR-команды

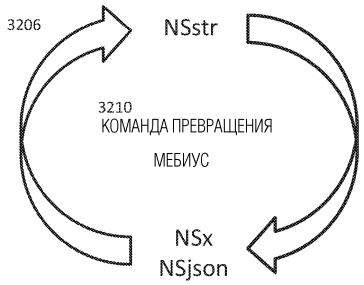
Фиг. 31

3202

ПРЕВРАЩЕНИЕ	ОПЕРАЦИЯ=	ВВОД	ВЫВОД	КОММЕНТАРИИ
МЕБИУС	nsjson	NSstr	NSjson	ДОЛЖНА БЫТЬ ПОСЛЕДНЕЙ TAR-КОМАНДОЙ

3204

ТИП СТРУКТУРЫ ВВОДА	NSX МЕБИУСА	ОБРАТНАЯ NSX МЕБИУСА	ВЫЗОВ ФУНКЦИИ МЕБИУСА
NSstr	NSx	ИГНОРИРОВАНИЕ	Н/Д
NSx	НЕВОЗМОЖНО, БЕЗ TAR	НЕВОЗМОЖНО, БЕЗ TAR	NSstr



Фиг. 32

3302

Превращение	Ввод	Вывод
прижатие	Конкретная для языка собственная структура данных	JSON-совместимая структура данных на языке
очистка	NSstr	NSstr (очищено)

3304

Превращение	Операция=	Прямой TAR	Обратный TAR	Комментарии
ключ	generate	Условный	Никогда	Автоматически вызывается при необходимости после "проверки ключей"
ключ	check	Автоматический	Автоматический	Пытаться сделать все возможное, чтобы выполнять успешную TAR

3310
КОМАНДЫ ПРЕВРАЩЕНИЯ

press
clean
key generate
key check

Фиг. 33

СЕКЦИЯ	ПОЛЕ	В	ФОРМИР.	ЗНАЧЕНИЯ	ОПРЕДЕЛЕНИЕ
typ		✓	✓	KISS	СПЕЦИФИКАТОР ТИПА СТРУКТУРЫ
id			✓	B64	ИДЕНТИФИКАТОР КЛЮЧА (PUID), NUTID, ЧТО УГОДНО, ЕСЛИ НИЧЕГО НЕТ
ima		✓	✓	ЗАМОЧНАЯ СКВАЖИНА, КЛЮЧ	УКАЗЫВАЕТ ТО, ЧТО ЭТОТ KES ДЕЛАЕТ ЗДЕСЬ. "Я – ЗАМОЧНАЯ СКВАЖИНА!"
key	v	✓	✓	ЗНАЧЕНИЕ(Я) КЛЮЧА	СТРОКА ИЛИ СПИСОК В ЗАВИСИМОСТИ ОТ КЛЮЧ->СЧЕТЧИК
	type	✓	✓	СИММЕТРИЧНЫЙ, RSA, DSA, ECC, ФРАЗОВЫЙ ПАРОЛЬ, IKM, SYMMETRICLIST, TINES255, TINESIDX128	ТИПЫ КЛЮЧЕЙ, ИСПОЛЬЗУЕМЫЕ В ШАБЛОНАХ КЛЮЧЕЙ
	format	✓	✓	B54, ШЕСТНАДЦАТЕРИЧНЫЙ, СТРОКА, PERN, ЭЛЕМЕНТ ВЫБОРКИ	СПОСОБ ФОРМАТА ДЛЯ ЗНАЧЕНИЯ KEV
	length		✓	ЦЕЛОЧИСЛЕННОЕ (СЧЕТЧИК БИТОВ/ СИМВОЛОВ)	ДЛИНА КЛЮЧА В БИТАХ
	part		✓	ЗАКРЫТЫЙ, ОТКРЫТЫЙ	ПРИМЕНЯЕТСЯ ТОЛЬКО К АСИММЕТРИЧНЫМ КЛЮЧАМ
	count		✓	ПО УМОЛЧАНИЮ = 1	ЕСЛИ>1, ТО ЗНАЧЕНИЕ КЛЮЧА ЯВЛЯЕТСЯ СПИСКОМ ЗНАЧЕНИЙ КЛЮЧА
	state		✓	ПОМЕЩ., ФОРМИР., ЗАПРАШИВАНИЕ, PUTOBJ	ОТКУДА ОНО ИСХОДИТ? ПОМЕЩ. (В), ФОРМИР(ОВАНИЕ) ИЛИ ЗАПРАШИВАНИЕ (ЕГО), PUTOBJ ИЗ ОДНОГО ВЫЗОВА ЧЕРЕЗ МЕТОД OBJ
salt	value		✓	ЗНАЧЕНИЕ СОЛИ	СОЛЬ, АССОЦИИРОВАННАЯ С ЭТИМ КЛЮЧОМ. ОНО МОЖЕТ ПРЕДОСТАВЛЯТЬСЯ ИЛИ ФОРМИРОВАТЬСЯ ПОСРЕДСТВОМ ЕГО ИСПОЛЬЗОВАНИЯ ПОСРЕДСТВОМ КРИПТОАЛГОРИТМА. ОНО МОЖЕТ БЫТЬ ИНФОРМАЦИОННЫМ ИЛИ ДОЛЖНО СОХРАНЯТЬСЯ ПОЛЬЗОВАТЕЛЕМ В ЗАВИСИМОСТИ ОТ ИСПОЛЬЗОВАНИЯ. В ОБЩЕМ, ДОЛЖНО СОХРАНЯТЬСЯ В РАСЧЕТЕ НА ЭЛЕМЕНТ ДАННЫХ, ПРИМЕНЯЕМЫЙ И НАХОДЯЩИХСЯ В НЕПОСРЕДСТВЕННОЙ БЛИЗОСТИ.
	type		✓	BEKT.ИН, ОДНОРАЗОВЫЙ НОМЕР, ИНДИВИДУАЛЬНО НАСТРАИВАЕМЫЙ	
	format		✓	B64, ШЕСТНАДЦАТЕРИЧНЫЙ, СТР., ЭЛЕМЕНТ ВЫБОРКИ	
	length		✓	ЦЕЛОЧИСЛЕННЫЙ (ЧИСЛО БИТОВ)	
	function		✓	СТРОКА	НАЗВАНИЕ ФУНКЦИИ, КОТОРАЯ ФОРМИРУЕТ СОЛЬ
	state		✓	ПОМЕЩЕН., ФОРМИР.	ОТКУДА ОНО ИСХОДИТ?
tag	label		✓		ОПИСАТЕЛЬНАЯ ИНФОРМАЦИЯ ДЛЯ КЛЮЧА
	description		✓		ОПИСАТЕЛЬНАЯ ИНФОРМАЦИЯ ДЛЯ КЛЮЧА
	question				ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ ДЛЯ ТИПА PP-КЛЮЧА
	hint				ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ ДЛЯ ТИПА PP-КЛЮЧА
	dign				ЦИФРОВАЯ ПОДПИСЬ КЛЮЧА ПОСРЕДСТВОМ ПРОИЗВОДИТЕЛЯ КЛЮЧЕЙ
dt	create		✓	СТРОКА В UTC-ФОРМАТЕ	ДАТА СОЗДАНИЯ КЛЮЧА
	madeby		✓		ИДЕНТИФИКАТОР СОЗДАТЕЛЯ
	active				ДАТА АКТИВАЦИИ KEV
	expire			СТРОКА В UTC-ФОРМАТЕ	ДАТА ИСТЕЧЕНИЯ СРОКА ДЕЙСТВИЯ КЛЮЧА
Могут добавляться другие секции, такие как сертификат, x.509 по мере необходимости. Поддержание зависит от каждой применимой программы					

Фиг. 34

3502

РЕЖИМ	ИСТОЧНИК	ПОСТОЯННОЕ ХРАНИЛИЩЕ	ПРИМЕНЯЕТСЯ К
ПРЕВРАЩЕНИЕ	ФОРМИРОВАНИЕ (ФОРМИР.)	ВСТРОЕННОЕ, ИНФОРМАЦИОННОЕ	ЗАШИФРОВАННЫЙ ТЕКСТ
	ВВОД (ПОМЕЩ.)		
ЗАМОЧНАЯ СКВАЖИНА	ФОРМИРОВАНИЕ (ФОРМИР.)	ОБЯЗАТЕЛЬНОЕ	
	ВВОД (ПОМЕЩ.)		ИЗВЛЕЧЕННЫЙ КЛЮЧ

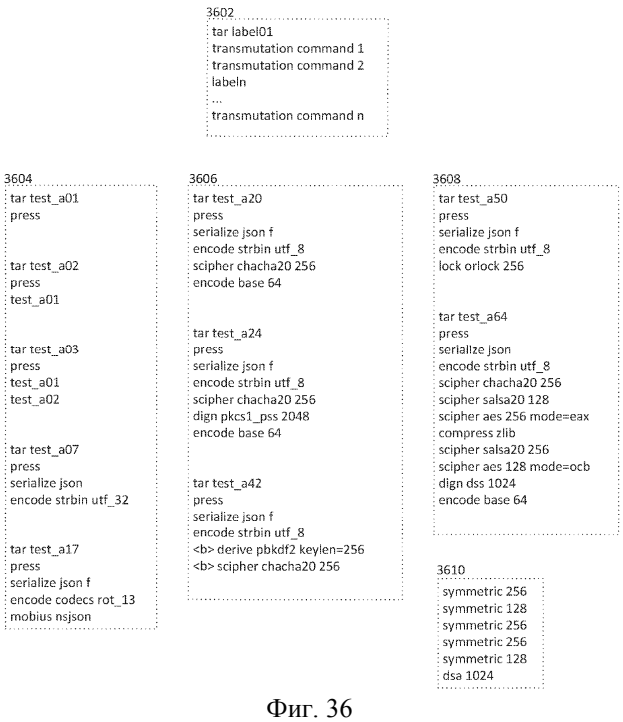
3504

КЛЮЧЕВЫЕ СЛОВА ПАРАМЕТРОВ ШАБЛОНОВ						ИСТОЧНИК СОЛИ		ХРАНИЛИЩЕ СОЛИ	
ТИП КЛЮЧА:	ДЛ. КЛЮЧА:	ЧИС. КЛЮЧЕЙ:	ПОРОГ. ЗНАЧ.:	ТИП СОЛИ:	ДЛ. СОЛИ	ПОМЕЩ.	ФОРМИР.	KISS	ВСТРОЕНН.
ФРАЗОВЫЙ ПАРОЛЬ	✓	×	×	✓		✓	✓	✓	×
СИММЕТРИЧНЫЙ	✓	×	×	✓		×	✓	×	✓
СИММЕТРИЧНЫЙ СПИСОК	✓	✓	×	✓		×	✓	×	✓
rsa	✓	×	×	×	✓*	×	×	×	✓*
dsa, ecc	✓	×	×	×		×	×	×	×
tines256, tinesidx128	✓	✓	✓	×		×	×	×	×

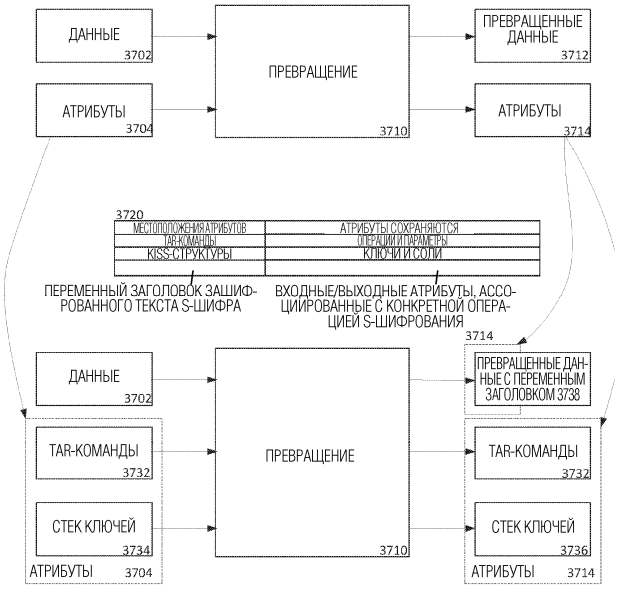
3506

ТИП КЛЮЧА:	ОПРЕДЕЛЕНИЕ
ФРАЗОВЫЙ ПАРОЛЬ	ЛЮБАЯ ПЕЧАТАЕМАЯ СТРОКА СИМВОЛОВ
ikm	МАТЕРИАЛ НАЧАЛЬНОГО КЛЮЧА
СИММЕТРИЧНЫЙ	КЛЮЧ ДЛЯ СИММЕТРИЧНОГО ШИФРА, ОБЫЧНО СЛУЧАЙНОЕ ЧИСЛО
СИММЕТРИЧНЫЙ СПИСОК	СПИСОК СИММЕТРИЧНЫХ КЛЮЧЕЙ
rsa	НАБОР АСИММЕТРИЧНЫХ КЛЮЧЕЙ ШИФРОВАНИЯ СОГЛАСНО RSA-СПОСОБОВ ДЛЯ RSA-ШИФРОВ И СХЕМ ПОДПИСИ
dsa	АЛГОРИТМ НА ОСНОВЕ ЦИФРОВОЙ ПОДПИСИ: НАБОР АСИММЕТРИЧНЫХ КЛЮЧЕЙ ДЛЯ DSS-СХЕМ ПОДПИСИ
ecc	НАБОР АСИММЕТРИЧНЫХ КЛЮЧЕЙ С ИСПОЛЬЗОВАНИЕМ СПОСОБОВ КРИПТОГРАФИИ В ЭЛЛИПТИЧЕСКИХ КРИВЫХ
tines256	СПИСОК КЛЮЧЕЙ ПО ПРИНЦИПУ РАЗДЕЛЕНИЯ СЕКРЕТОВ ДЛИНОЙ В 256 БИТОВ СО ВСТРОЕННЫМИ ИНДЕКСАМИ
tinesidx128	СПИСОК КЛЮЧЕЙ ПО ПРИНЦИПУ РАЗДЕЛЕНИЯ СЕКРЕТОВ ДЛИНОЙ В 128 БИТОВ С ДОБАВЛЕННЫМИ В НАЧАЛО 128-БИТОВЫМИ ИНДЕКСАМИ

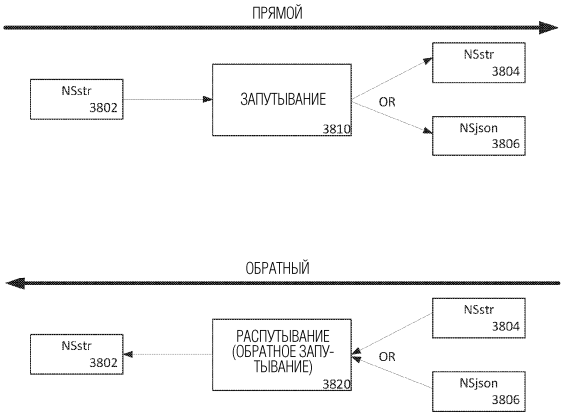
Фиг. 35



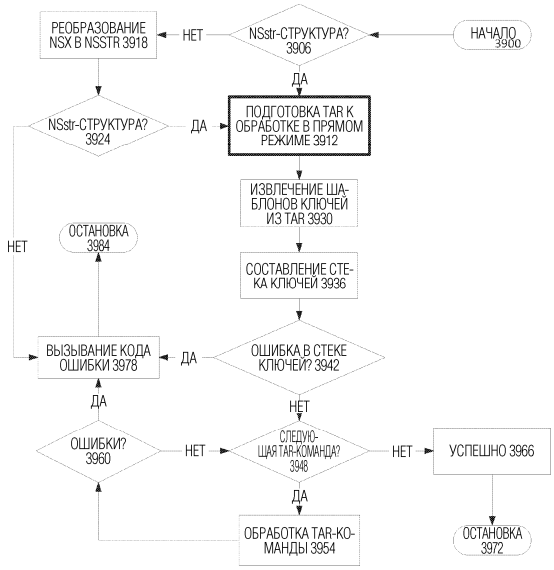
Фиг. 36



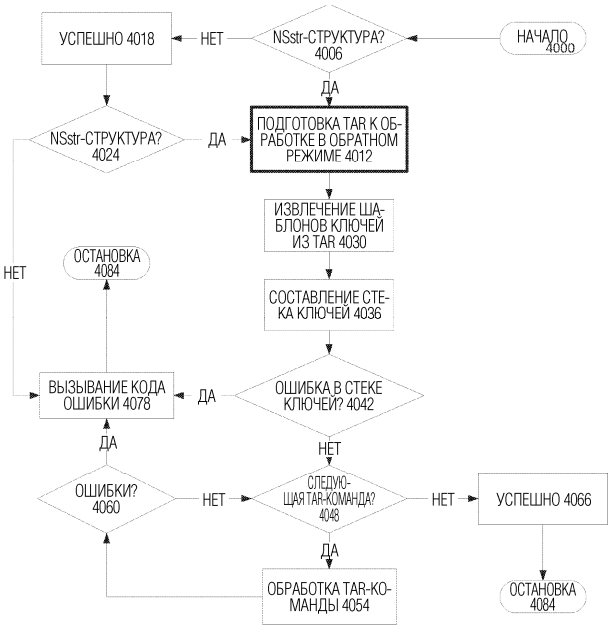
Фиг. 37



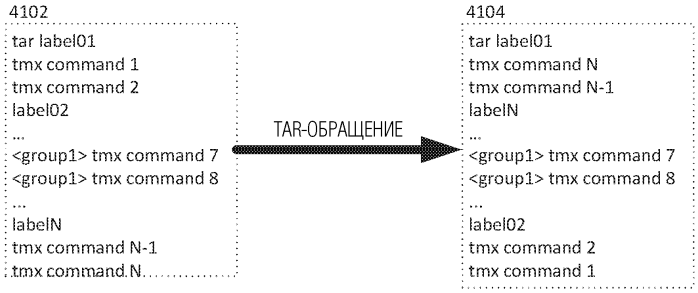
Фиг. 38



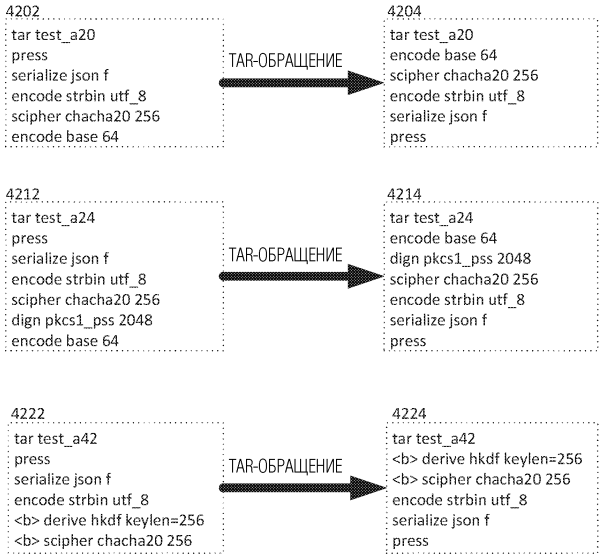
Фиг. 39



Фиг. 40



Фиг. 41



Фиг. 42

4300

ПРЕВРАЩЕНИЕ (ТИП ОПЕРАЦИИ)	ШАБЛОН ТИПОВ КЛЮЧЕЙ
ИЗВЛЕЧЕНИЕ [pbkdf2, scrypt]	ФРАЗОВЫЙ ПАРОЛЬ
ИЗВЛЕЧЕНИЕ hkdf	ikm
S-ШИФР	СИММЕТРИЧНЫЙ
S-ШИФР	rsa
Ц-ПОДПИСЬ [pkcs1_pss, pkcs1_v1_5]	rsa
Ц-ПОДПИСЬ dss [1024, 2048, 3072]	dsa
Ц-ПОДПИСЬ dss 256	ecc
ЗАМОК «OR-ЗАМОК»	СИММЕТРИЧНЫЙ
ЗАМОК «OR-ЗАМОК»	СИММЕТРИЧНЫЙ СПИСОК
ЗАМОК (МАТН-ЗАМОК, XOR-ЗАМОК, ХЭШ-ЗАМОК)	СИММЕТРИЧНЫЙ СПИСОК
ЗАМОК «SS-ЗАМОК»	tines256
ЗАМОК «SS-ЗАМОК_B»	tinesidx128

Фиг. 43

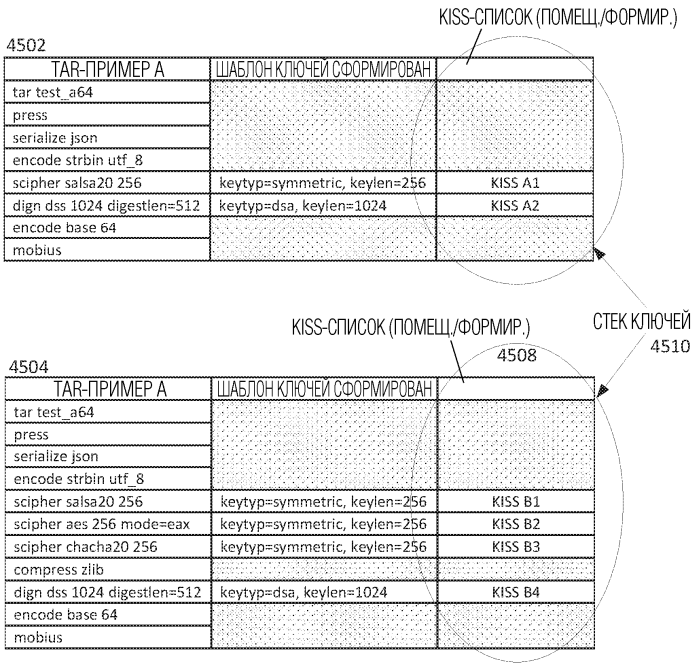
4402

TAR-ПРИМЕР A	ШАБЛОН КЛЮЧЕЙ СФОРМИРОВАН
tar test_a64	
press	
serialize json	
encode strbin utf_8	
scipher salsa20 256	keytyp=symmetric, keylen=256
dign dss 1024 digestlen=512	keytyp=dsa, keylen=1024
encode base 64	
mobius	

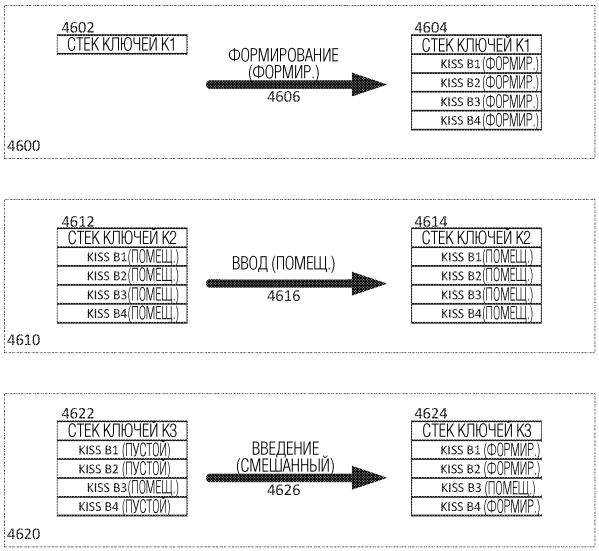
4404

TAR-ПРИМЕР B	ШАБЛОН КЛЮЧЕЙ СФОРМИРОВАН
tar test_a64	
press	
serialize json	
encode strbin utf_8	
scipher salsa20 256	keytyp=symmetric, keylen=256
scipher aes 256 mode=eax	keytyp=symmetric, keylen=256
scipher chacha20 256	keytyp=symmetric, keylen=256
compress zlib	
dign dss 1024 digestlen=512	keytyp=dsa, keylen=1024
encode base 64	
mobius	

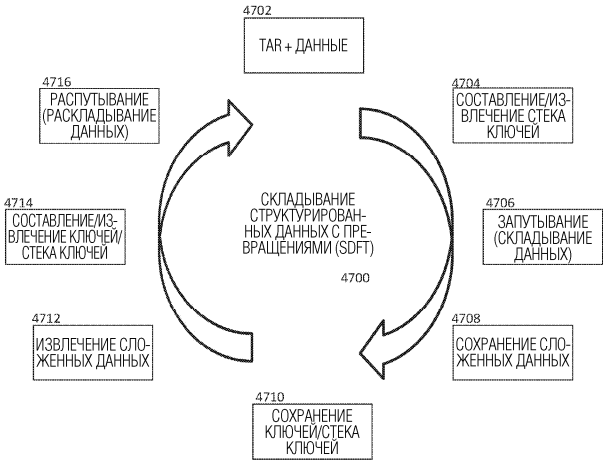
Фиг. 44



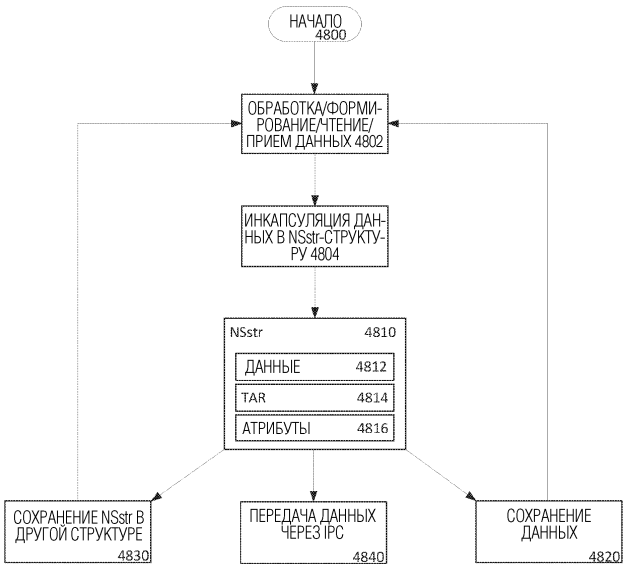
Фиг. 45



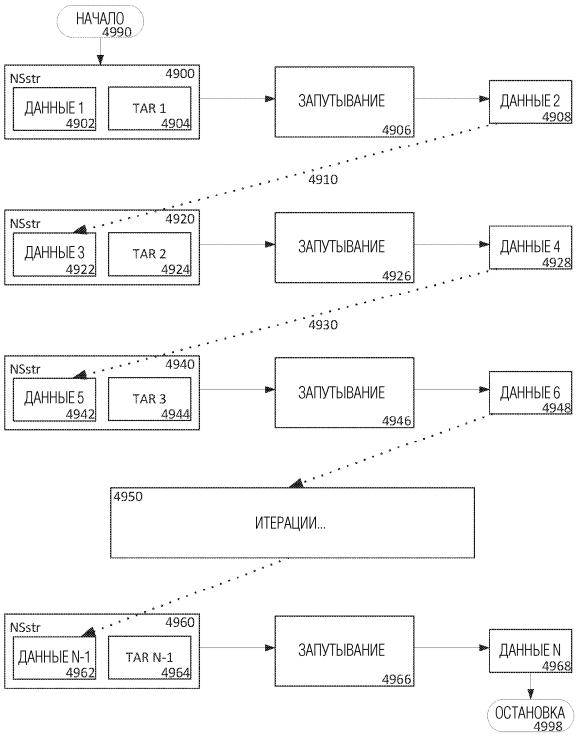
Фиг. 46



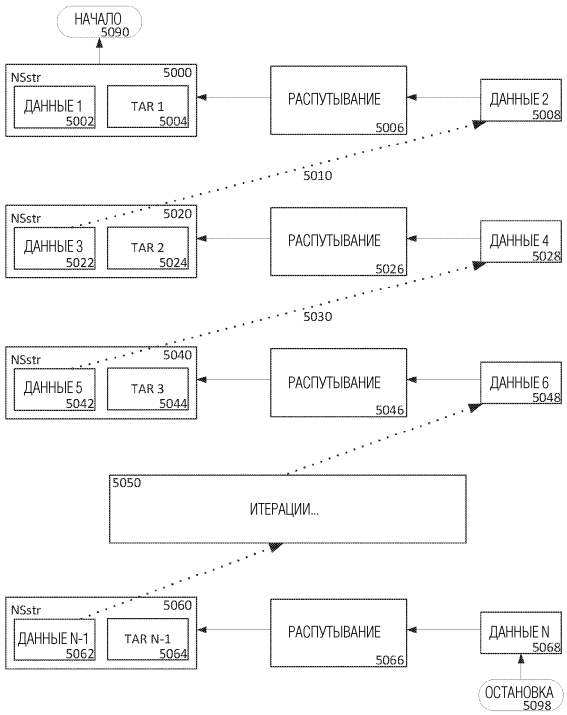
Фиг. 47



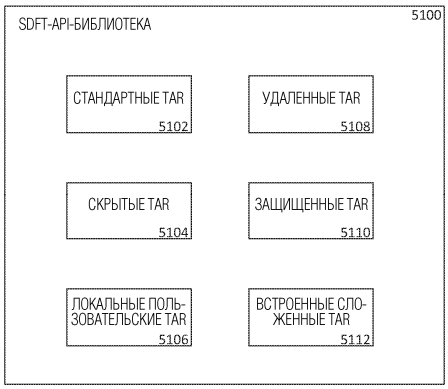
Фиг. 48


НАЧАЛО
4990
NSstr
4900
ДАННЫЕ 1
4902
TAR 1
4904
ЗАПУТЫВАНИЕ
4906
ДАННЫЕ 2
4908
4910
NSstr
4920
ДАННЫЕ 3
4922
TAR 2
4924
ЗАПУТЫВАНИЕ
4926
ДАННЫЕ 4
4928
4930
NSstr
4940
ДАННЫЕ 5
4942
TAR 3
4944
ЗАПУТЫВАНИЕ
4946
ДАННЫЕ 6
4948
4950
ИТЕРАЦИИ...
NSstr
4960
ДАННЫЕ N-1
4962
TAR N-1
4964
ЗАПУТЫВАНИЕ
4966
ДАННЫЕ N
4968
ОСТАНОВКА
4998

Фиг. 49


НАЧАЛО
5090
NSstr
5000
ДАННЫЕ 1
5002
TAR 1
5004
РАСПУТЫВАНИЕ
5006
ДАННЫЕ 2
5008
5010
NSstr
5020
ДАННЫЕ 3
5022
TAR 2
5024
РАСПУТЫВАНИЕ
5026
ДАННЫЕ 4
5028
5030
NSstr
5040
ДАННЫЕ 5
5042
TAR 3
5044
РАСПУТЫВАНИЕ
5046
ДАННЫЕ 6
5048
5050
ИТЕРАЦИИ...
NSstr
5060
ДАННЫЕ N-1
5062
TAR N-1
5064
РАСПУТЫВАНИЕ
5066
ДАННЫЕ N
5068
ОСТАНОВКА
5098

Фиг. 50



Фиг. 51

5210

```
СТРОКА 01  data = dict(string = 'oops', bytes = b'234', deeper = ['str2', b'2']}
```

5220

```
СТРОКА 02  ЗАДАНИЕ ДАННЫХ JSON-СОВМЕСТИМЫМИ ПОСРЕДСТВОМ ПРЕОБРАЗОВАНИЯ ВСЕХ БАЙТОВ В ДАННЫЕ
СТРОКА 03  ВЫПОЛНЕНИЕ JSON-ПРЕОБРАЗОВАНИЯ В ПОСЛЕДОВАТЕЛЬНУЮ ФОРМУ ДЛЯ ДАННЫХ
СТРОКА 04  ПРЕОБРАЗОВАНИЕ JSON-СТРОКИ В БАЙТОВУЮ СТРОКУ
СТРОКА 05  ВЫЧИСЛЕНИЕ CRC 16-ХЭША ДЛЯ ДАННЫХ
СТРОКА 06  ОБЕРТЫВАНИЕ ДАННЫХ И ДАЙДЖЕСТА В СТРУКТУРУ
СТРОКА 07  ЗАПИСЬ ДАННЫХ В ФАЙЛ
```

5250

```
СТРОКА 08  #ВЫПОЛНЕНИЕ СКЛАДЫВАНИЯ ДАННЫХ ВРУЧНУЮ (ЗАПУТЫВАНИЯ)
СТРОКА 09  import base64, json, binascii
СТРОКА 10  data['bytes'] = base64.b64encode(data['bytes']).decode()
СТРОКА 11  data['deeper'][1] = base64.b64encode(data['deeper'][1]).decode()
СТРОКА 12  data = json.dumps(data, sort_keys=False)
СТРОКА 13  data = data.encode('utf_8')
СТРОКА 14  digest = binascii.crc_hqx(data,0).to_bytes(2, byteorder = 'big')
СТРОКА 15  wrap = dict(data=data.decode(), digest=base64.b64encode(digest).decode())
СТРОКА 16  with open("mydata.json", "w") as f:
               json.dump(wrap, f)
```

5260

```
СТРОКА 17  #ВЫПОЛНЕНИЕ РАСКЛАДЫВАНИЯ ДАННЫХ ВРУЧНУЮ
СТРОКА 18  import base64, json
СТРОКА 19  with open("mydata.json", "r") as f:
               wrap = json.load(f)
СТРОКА 20  digest = binascii.crc_hqx(wrap['data'].encode(),0).to_bytes(2, byteorder = 'big')
СТРОКА 21  if digest != base64.b64decode(wrap['digest']):
               print('error: crc codes do not match')
СТРОКА 22  data = json.loads(wrap['data'])
СТРОКА 23  data['deeper'][1] = base64.b64decode(data['deeper'][1])
СТРОКА 24  data['bytes'] = base64.b64decode(data['bytes'])
СТРОКА 25
```

Фиг. 52

5310

```
СТРОКА 01  data = dict(string = 'oops', bytes = b'234', deeper = ['str2', b'2']}
```

5320

```
СТРОКА 02  tar test_a70
СТРОКА 03  press
СТРОКА 04  serialize json f
СТРОКА 05  encode strbin utf_8
СТРОКА 06  digest hash crc 16
СТРОКА 07  encode base 64
```

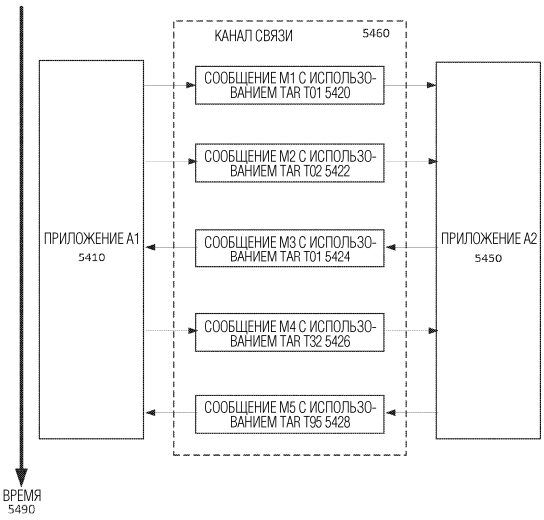
5350

```
СТРОКА 08  #ВЫПОЛНЕНИЕ ЗАПУТЫВАНИЯ ЧЕРЕЗ SDFT
СТРОКА 09  import NSsdf
СТРОКА 10  ns = NSsdf.NSstring(data)
СТРОКА 11  retobj = ns.ravel(tarName="test_a70")
           ns.writeJSONfile("mydata.json")
```

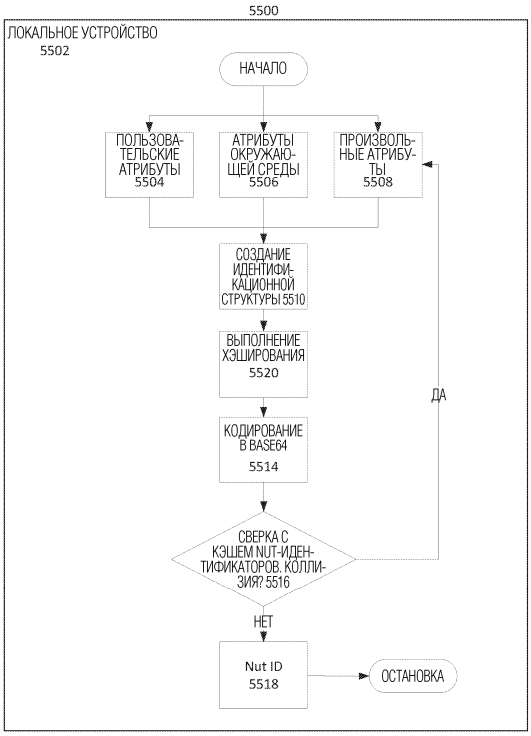
5360

```
СТРОКА 12  #ВЫПОЛНЕНИЕ РАСПУТЫВАНИЯ ЧЕРЕЗ SDFT
СТРОКА 13  import NSutil, NSsdf
СТРОКА 14  ns = NSsdf.NSstring(NSutil.readJSONfile("mydata.json"))
СТРОКА 15  retobj = ns.unravel()
           data = ns.getObj()
```

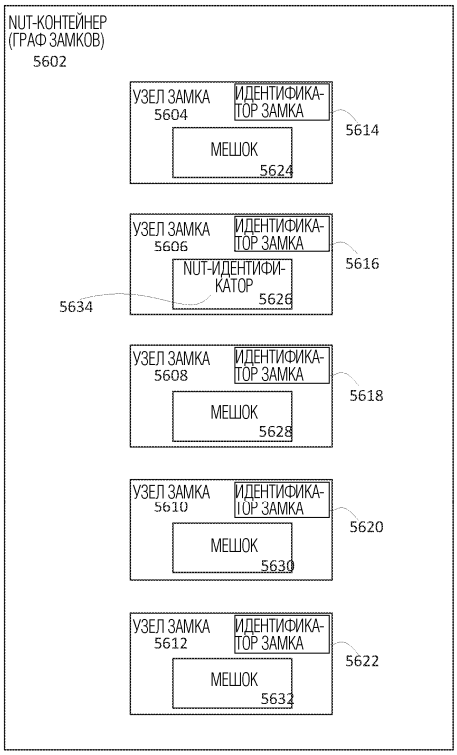
Фиг. 53


КАНАЛ СВЯЗИ
5460
ПРИЛОЖЕНИЕ А1
5410
ПРИЛОЖЕНИЕ А2
5450
СООБЩЕНИЕ M1 С ИСПОЛЬЗОВАНИЕМ TAR T01 5420
СООБЩЕНИЕ M2 С ИСПОЛЬЗОВАНИЕМ TAR T02 5422
СООБЩЕНИЕ M3 С ИСПОЛЬЗОВАНИЕМ TAR T01 5424
СООБЩЕНИЕ M4 С ИСПОЛЬЗОВАНИЕМ TAR T32 5426
СООБЩЕНИЕ M5 С ИСПОЛЬЗОВАНИЕМ TAR T95 5428
ВРЕМЯ
5490

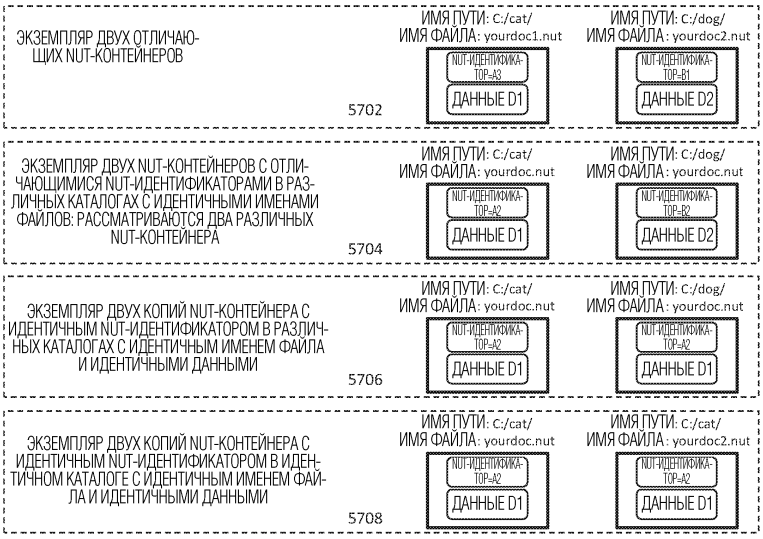
Фиг. 54


5500
ЛОКАЛЬНОЕ УСТРОЙСТВО
5502
НАЧАЛО
ПОЛЬЗОВАТЕЛЬСКИЕ АТРИБУТЫ
5504
АТРИБУТЫ ОКРУЖАЮЩЕЙ СРЕДЫ
5506
ПРОИЗВОЛЬНЫЕ АТРИБУТЫ
5508
СОЗДАНИЕ ИДЕНТИФИКАЦИОННОЙ СТРУКТУРЫ 5510
ВЫПОЛНЕНИЕ ХЭШИРОВАНИЯ
5520
КОДИРОВАНИЕ В BASE64
5514
СВЕРКА С КЭШЕМ NUT-ИДЕНТИФИКАТОРОВ. КОЛЛИЗИЯ? 5516
ДА
НЕТ
Nut ID
5518
ОСТАНОВКА

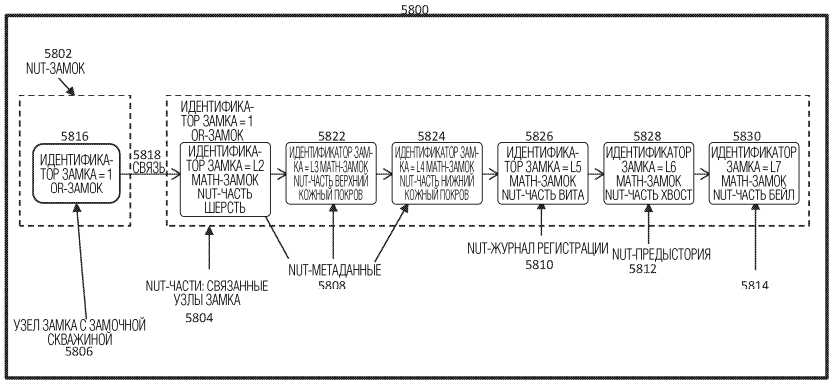
Фиг. 55



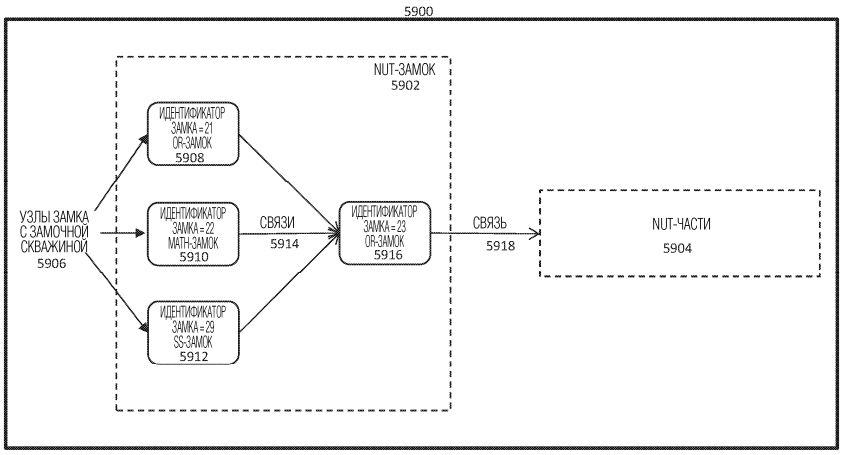
Фиг. 56



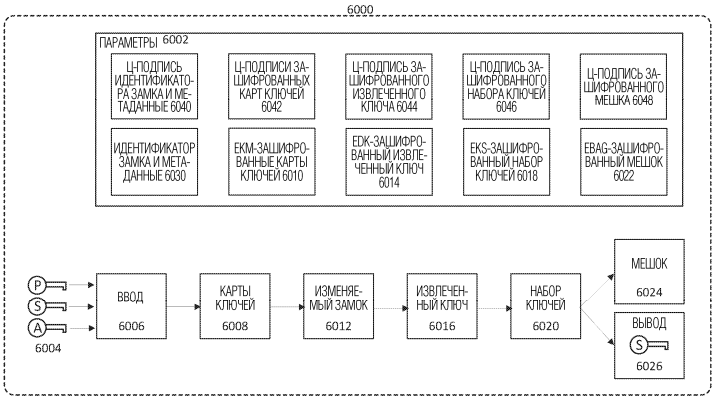
Фиг. 57



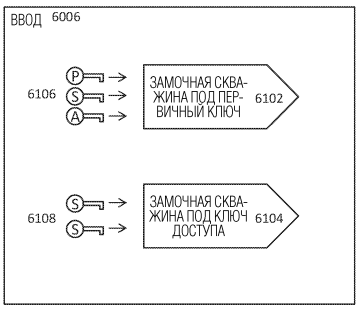
Фиг. 58



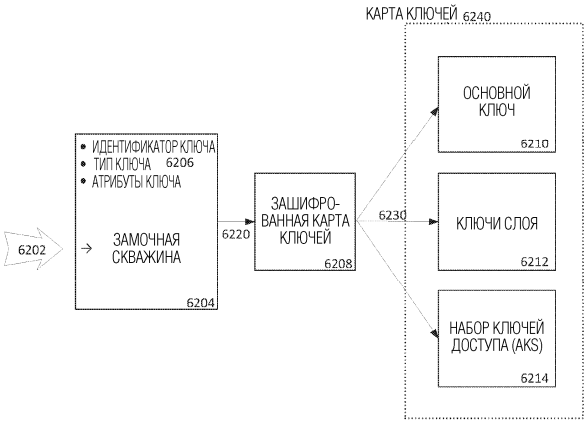
Фиг. 59



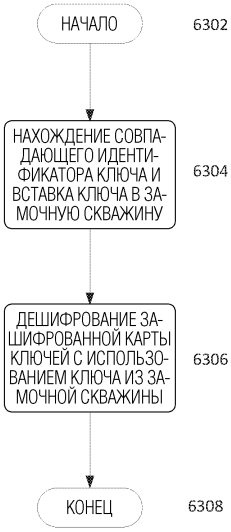
Фиг. 60



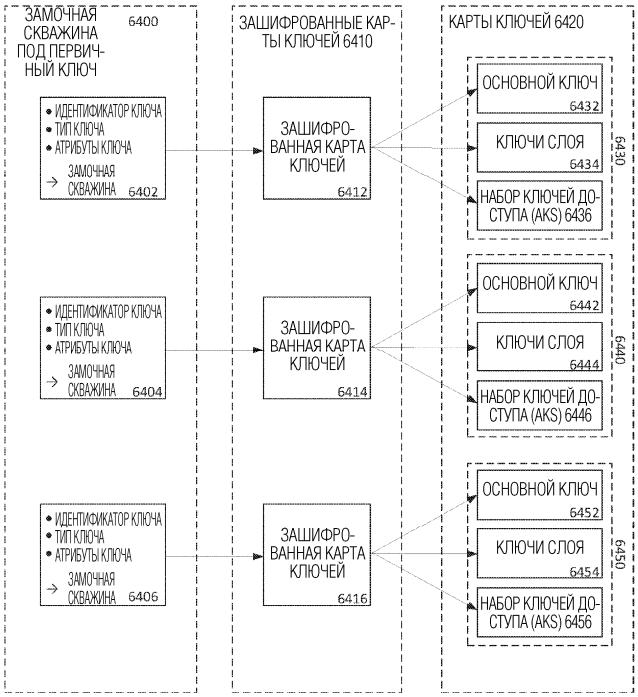
Фиг. 61



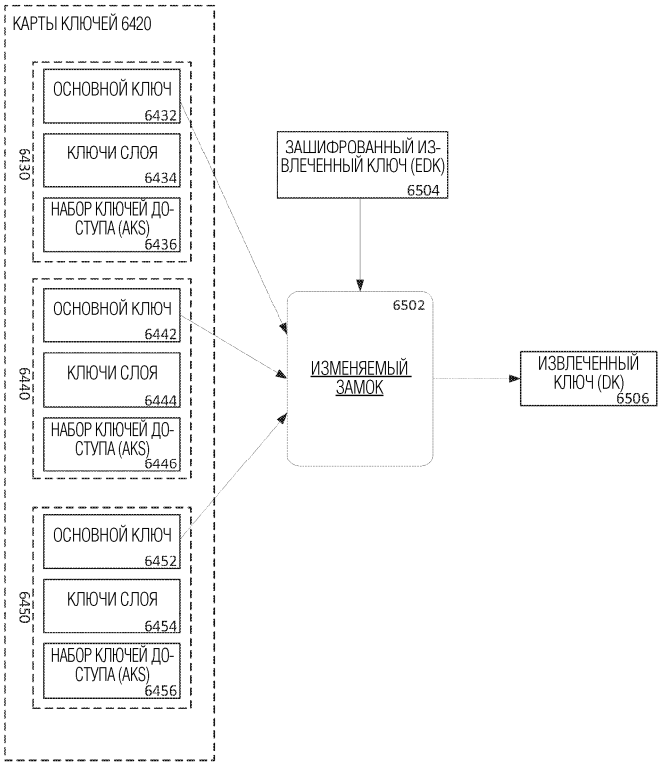
Фиг. 62


НАЧАЛО
6302
НАХОЖДЕНИЕ СОВПАДАЮЩЕГО ИДЕНТИФИКАТОРА КЛЮЧА И ВСТАВКА КЛЮЧА В ЗАМОЧНУЮ СКВАЖИНУ
6304
ДЕШИФРОВАНИЕ ЗАШИФРОВАННОЙ КАРТЫ КЛЮЧЕЙ С ИСПОЛЬЗОВАНИЕМ КЛЮЧА ИЗ ЗАМОЧНОЙ СКВАЖИНЫ
6306
КОНЕЦ
6308

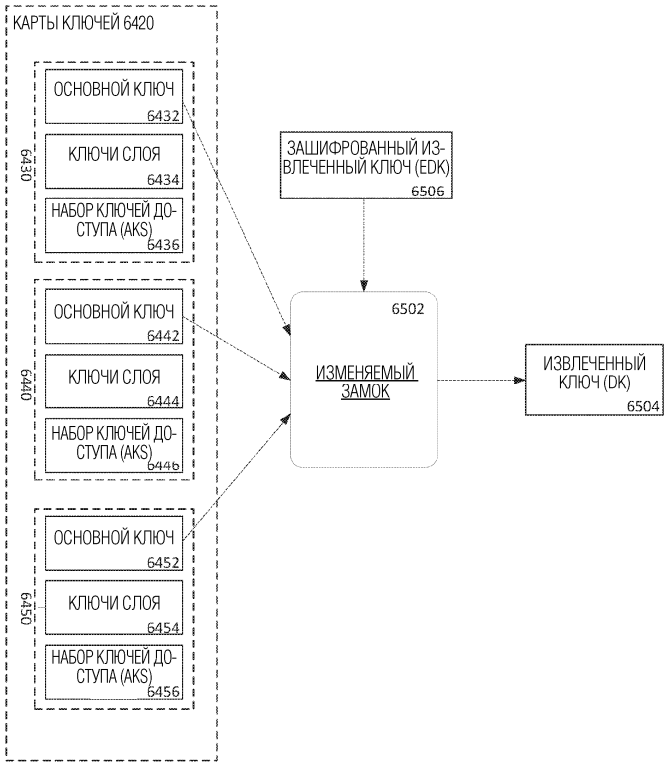
Фиг. 63


ЗАМОЧНАЯ СКВАЖИНА ПОД ПЕРВИЧНЫЙ КЛЮЧ
6400
ЗАШИФРОВАННЫЕ КАРТЫ КЛЮЧЕЙ 6410
КАРТЫ КЛЮЧЕЙ 6420
ИДЕНТИФИКАТОР КЛЮЧА
ТИП КЛЮЧА
АТРИБУТЫ КЛЮЧА
ЗАМОЧНАЯ СКВАЖИНА
6402
ЗАШИФРОВАННАЯ КАРТА КЛЮЧЕЙ
6412
ОСНОВНОЙ КЛЮЧ
6432
КЛЮЧИ СЛОЯ
6434
НАБОР КЛЮЧЕЙ ДОСТУПА (AKS) 6436
6430
ИДЕНТИФИКАТОР КЛЮЧА
ТИП КЛЮЧА
АТРИБУТЫ КЛЮЧА
ЗАМОЧНАЯ СКВАЖИНА
6404
ЗАШИФРОВАННАЯ КАРТА КЛЮЧЕЙ
6414
ОСНОВНОЙ КЛЮЧ
6442
КЛЮЧИ СЛОЯ
6444
НАБОР КЛЮЧЕЙ ДОСТУПА (AKS) 6446
6440
ИДЕНТИФИКАТОР КЛЮЧА
ТИП КЛЮЧА
АТРИБУТЫ КЛЮЧА
ЗАМОЧНАЯ СКВАЖИНА
6406
ЗАШИФРОВАННАЯ КАРТА КЛЮЧЕЙ
6416
ОСНОВНОЙ КЛЮЧ
6452
КЛЮЧИ СЛОЯ
6454
НАБОР КЛЮЧЕЙ ДОСТУПА (AKS) 6456
6450

Фиг. 64


КАРТЫ КЛЮЧЕЙ 6420
6430
ОСНОВНОЙ КЛЮЧ 6432
КЛЮЧИ СЛОЯ 6434
НАБОР КЛЮЧЕЙ ДОСТУПА (AKS) 6436
6440
ОСНОВНОЙ КЛЮЧ 6442
КЛЮЧИ СЛОЯ 6444
НАБОР КЛЮЧЕЙ ДОСТУПА (AKS) 6446
6450
ОСНОВНОЙ КЛЮЧ 6452
КЛЮЧИ СЛОЯ 6454
НАБОР КЛЮЧЕЙ ДОСТУПА (AKS) 6456
ЗАШИФРОВАННЫЙ ИЗВЛЕЧЕННЫЙ КЛЮЧ (EDK) 6504
6502
ИЗМЕНЯЕМЫЙ ЗАМОК
ИЗВЛЕЧЕННЫЙ КЛЮЧ (DK) 6506

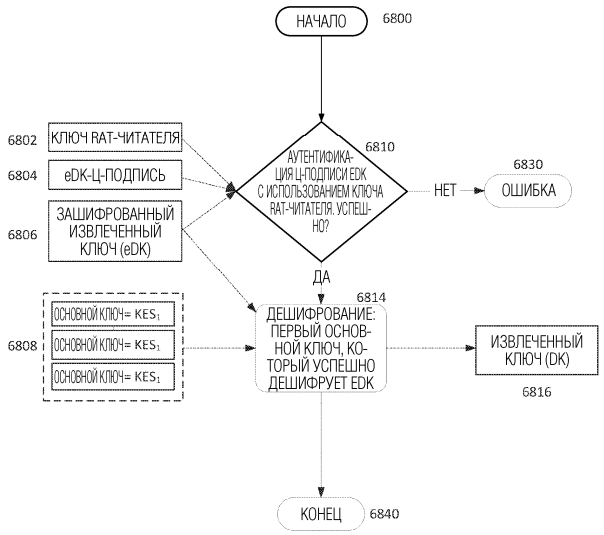
Фиг. 65


КАРТЫ КЛЮЧЕЙ 6420
6430
ОСНОВНОЙ КЛЮЧ 6432
КЛЮЧИ СЛОЯ 6434
НАБОР КЛЮЧЕЙ ДОСТУПА (AKS) 6436
6440
ОСНОВНОЙ КЛЮЧ 6442
КЛЮЧИ СЛОЯ 6444
НАБОР КЛЮЧЕЙ ДОСТУПА (AKS) 6446
6450
ОСНОВНОЙ КЛЮЧ 6452
КЛЮЧИ СЛОЯ 6454
НАБОР КЛЮЧЕЙ ДОСТУПА (AKS) 6456
ЗАШИФРОВАННЫЙ ИЗВЛЕЧЕННЫЙ КЛЮЧ (EDK) 6506
6502
ИЗМЕНЯЕМЫЙ ЗАМОК
ИЗВЛЕЧЕННЫЙ КЛЮЧ (DK) 6504

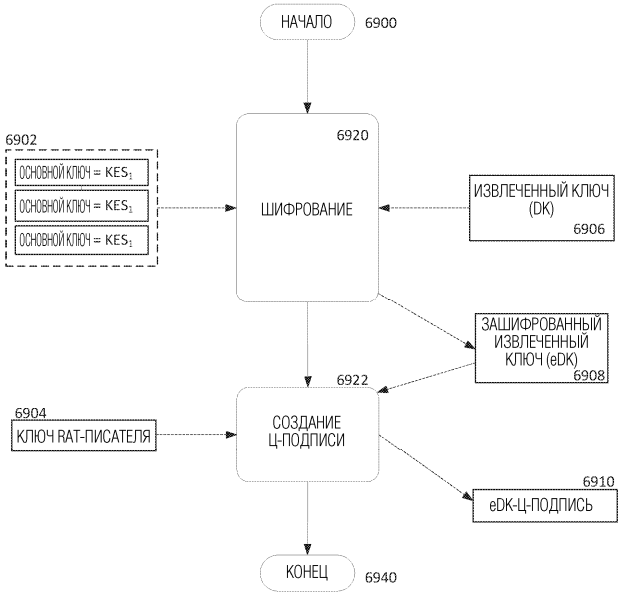
Фиг. 66

ТИП ИЗМЕНЯЕМЫХ ЗАМКОВ	ПОДТВЕРЖДЕННЫЕ КЛЮЧИ	НЕОБХОДИМЫЕ КЛЮЧИ	ТИП КЛЮЧА	ОПИСАНИЕ
OR-ЗАМОК	k:{l,n}	1	СИММЕТРИЧНЫЙ	ЛЮБОЙ ОДИН ДОПУСТИМЫЙ КЛЮЧ ДОЛЖЕН ОТКРЫВАТЬ ЭТОТ ЗАМОК. ПРЕДУСМОТРЕНО N ЗАРЕГИСТРИРОВАННЫХ КЛЮЧЕЙ.
MAT-ЗАМОК	k:{n}	n	СИММЕТРИЧНЫЙ	КАЖДЫЙ КЛЮЧ ИСПОЛЬЗУЕТСЯ В ОТСОРТИРОВАННОМ ПОРЯДКЕ. ПРЕДУСМОТРЕНО N ЗАРЕГИСТРИРОВАННЫХ КЛЮЧЕЙ.
XOR-ЗАМОК	k:{n}	n>l	СИММЕТРИЧНЫЙ	ВСЕ КЛЮЧИ ПОДВЕРГАЮТСЯ XOR-ОПЕРАЦИИ, ЧТОБЫ СОЗДАВАТЬ ВЫЧИСЛЕННЫЙ КЛЮЧ. ПРЕДУСМОТРЕНО N ЗАРЕГИСТРИРОВАННЫХ КЛЮЧЕЙ.
ХЭШ-ЗАМОК	k:{n}	n	СИММЕТРИЧНЫЙ	ВСЕ КЛЮЧИ КОНКАТЕНИРУЮТСЯ В ОТСОРТИРОВАННОМ ПОРЯДКЕ И ХЭШИРУЮТСЯ, ЧТОБЫ СОЗДАВАТЬ ВЫЧИСЛЕННЫЙ КЛЮЧ. ПРЕДУСМОТРЕНО N ЗАРЕГИСТРИРОВАННЫХ КЛЮЧЕЙ.
SS-ЗАМОК	k:{m,n}	1>m<=k<=n	ЗУБЕЦ	ЗАМОК ПО ПРИНЦИПУ РАЗДЕЛЕНИЯ СЕКРЕТОВ, В КОТОРОМ ПРЕДУСМОТРЕНО N КВОТ И ПОРОГОВОЕ ЗНАЧЕНИЕ M КВОТ.

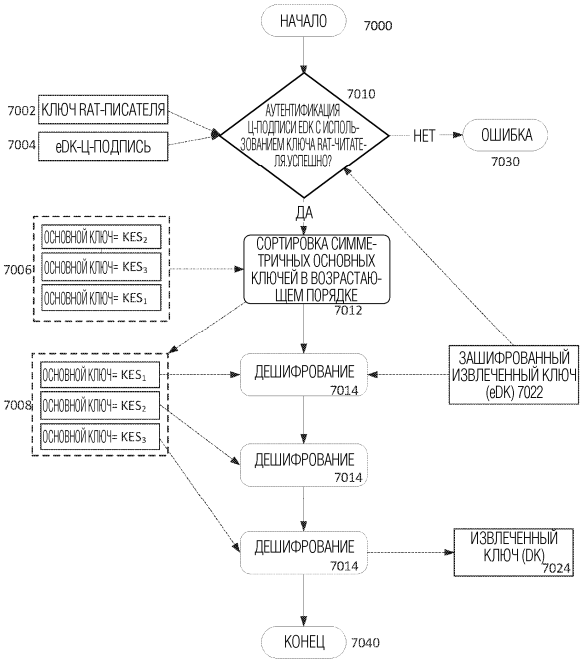
Фиг. 67



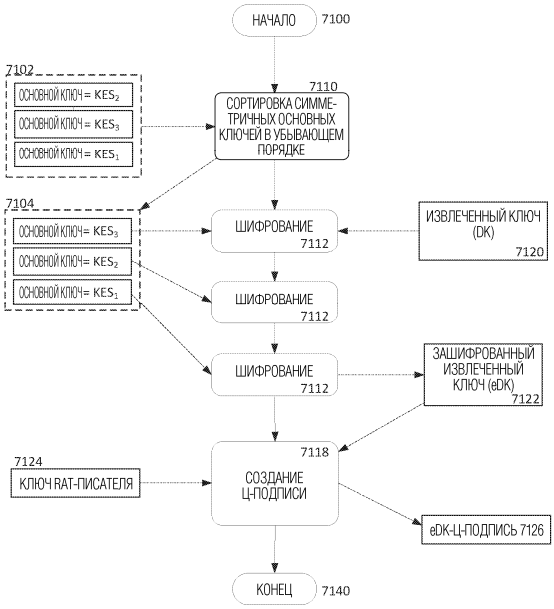
Фиг. 68



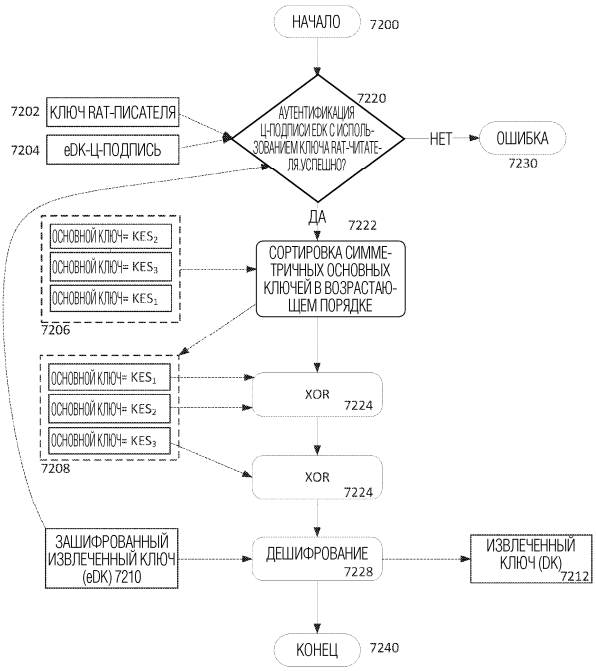
Фиг. 69


НАЧАЛО
7000
7002
КЛЮЧ RAT-ПИСАТЕЛЯ
7004
eDK-Ц-ПОДПИСЬ
АУТЕНТИФИКАЦИЯ Ц-ПОДПИСИ EDK С ИСПОЛЬЗОВАНИЕМ КЛЮЧА RAT-ЧИТАТЕЛЯ УСПЕШНО?
7010
НЕТ
ОШИБКА
7030
ДА
7006
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
ОСНОВНОЙ КЛЮЧ= KES1
СОРТИРОВКА СИММЕТРИЧНЫХ ОСНОВНЫХ КЛЮЧЕЙ В ВОЗРАСТАЮЩЕМ ПОРЯДКЕ
7012
7008
ОСНОВНОЙ КЛЮЧ= KES1
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
ДЕШИФРОВАНИЕ
7014
ЗАШИФРОВАННЫЙ ИЗВЛЕЧЕННЫЙ КЛЮЧ (eDK) 7022
ДЕШИФРОВАНИЕ
7014
ДЕШИФРОВАНИЕ
7014
ИЗВЛЕЧЕННЫЙ КЛЮЧ (DK)
7024
КОНЕЦ
7040

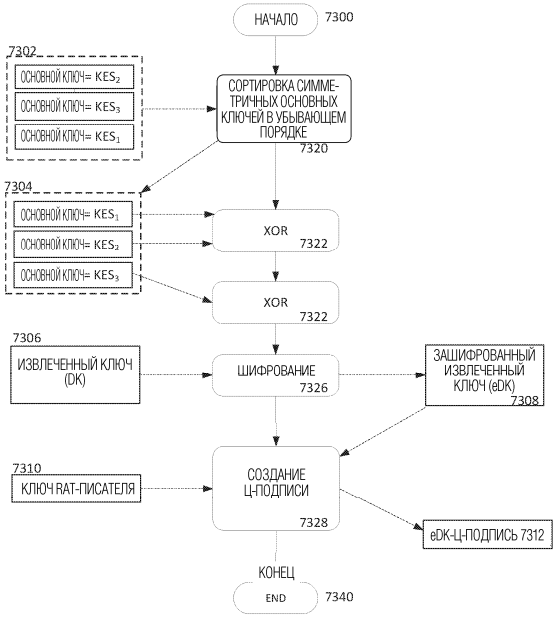
Фиг. 70


НАЧАЛО
7100
7102
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
ОСНОВНОЙ КЛЮЧ= KES1
7110
СОРТИРОВКА СИММЕТРИЧНЫХ ОСНОВНЫХ КЛЮЧЕЙ В УБЫВАЮЩЕМ ПОРЯДКЕ
7104
ОСНОВНОЙ КЛЮЧ= KES3
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES1
ШИФРОВАНИЕ
7112
ИЗВЛЕЧЕННЫЙ КЛЮЧ (DK)
7120
ШИФРОВАНИЕ
7112
ШИФРОВАНИЕ
7112
ЗАШИФРОВАННЫЙ ИЗВЛЕЧЕННЫЙ КЛЮЧ (eDK)
7122
7118
7124
КЛЮЧ RAT-ПИСАТЕЛЯ
СОЗДАНИЕ Ц-ПОДПИСИ
eDK-Ц-ПОДПИСЬ 7126
КОНЕЦ
7140

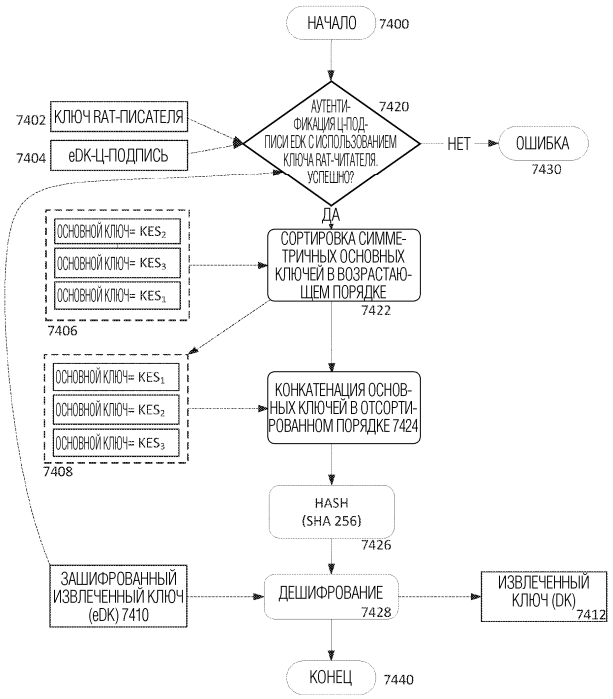
Фиг. 71


НАЧАЛО
7200
7202
КЛЮЧ RAT-ПИСАТЕЛЯ
7204
eDK-Ц-ПОДПИСЬ
АУТЕНТИФИКАЦИЯ Ц-ПОДПИСИ ЕДК С ИСПОЛЬЗОВАНИЕМ КЛЮЧА RAT-ЧИТАТЕЛЯ УСПЕШНО?
7220
НЕТ
ОШИБКА
7230
ДА
7222
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
ОСНОВНОЙ КЛЮЧ= KES1
7206
СОРТИРОВКА СИММЕТРИЧНЫХ ОСНОВНЫХ КЛЮЧЕЙ В ВОЗРАСТАЮЩЕМ ПОРЯДКЕ
ОСНОВНОЙ КЛЮЧ= KES1
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
7208
XOR
7224
XOR
7224
ЗАШИФРОВАННЫЙ ИЗВЛЕЧЕННЫЙ КЛЮЧ (eDK) 7210
ДЕШИФРОВАНИЕ
7228
ИЗВЛЕЧЕННЫЙ КЛЮЧ (DK)
7212
КОНЕЦ
7240

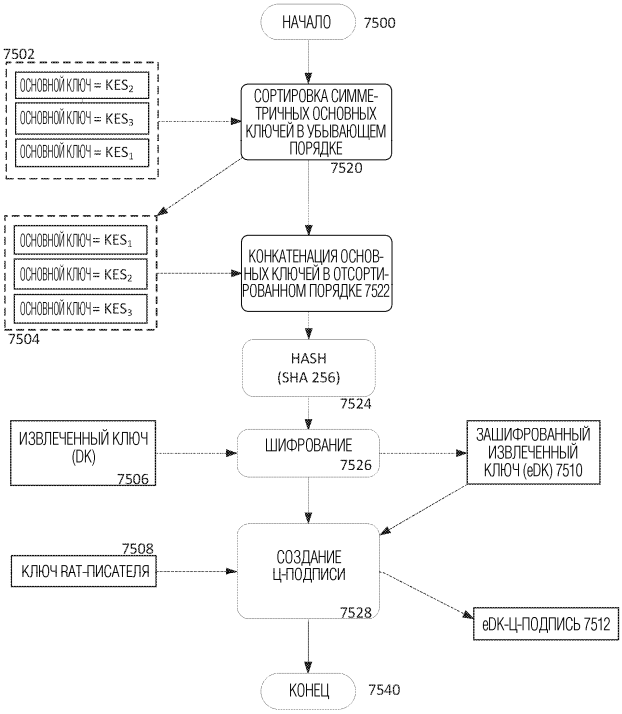
Фиг. 72


НАЧАЛО
7300
7302
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
ОСНОВНОЙ КЛЮЧ= KES1
СОРТИРОВКА СИММЕТРИЧНЫХ ОСНОВНЫХ КЛЮЧЕЙ В УБЫВАЮЩЕМ ПОРЯДКЕ
7320
7304
ОСНОВНОЙ КЛЮЧ= KES1
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
XOR
7322
XOR
7322
7306
ИЗВЛЕЧЕННЫЙ КЛЮЧ (DK)
ШИФРОВАНИЕ
7326
ЗАШИФРОВАННЫЙ ИЗВЛЕЧЕННЫЙ КЛЮЧ (eDK)
7308
7310
КЛЮЧ RAT-ПИСАТЕЛЯ
СОЗДАНИЕ Ц-ПОДПИСИ
7328
eDK-Ц-ПОДПИСЬ 7312
КОНЕЦ
END
7340

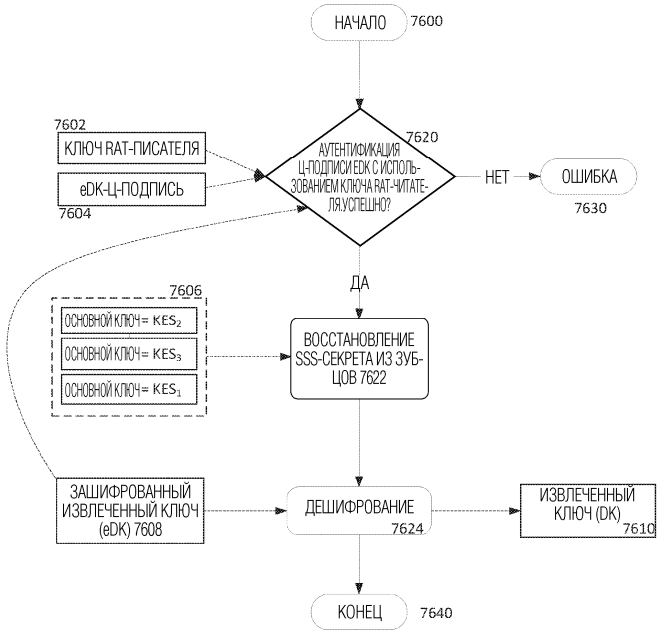
Фиг. 73


НАЧАЛО
7400
7402
КЛЮЧ RAT-ПИСАТЕЛЯ
7404
eDK-Ц-ПОДПИСЬ
АУТЕНТИ-ФИКАЦИЯ Ц-ПОД-ПИСИ EDK С ИСПОЛЬЗОВАНИЕМ КЛЮЧА RAT-ЧИТАТЕЛЯ. УСПЕШНО?
7420
НЕТ
ОШИБКА
7430
ДА
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
ОСНОВНОЙ КЛЮЧ= KES1
7406
СОРТИРОВКА СИММЕ-ТРИЧНЫХ ОСНОВНЫХ КЛЮЧЕЙ В ВОЗРАСТАЮ-ЩЕМ ПОРЯДКЕ
7422
ОСНОВНОЙ КЛЮЧ= KES1
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
7408
КОНКАТЕНАЦИЯ ОСНОВ-НЫХ КЛЮЧЕЙ В ОТСОРТИ-РОВАННОМ ПОРЯДКЕ 7424
HASH (SHA 256)
7426
ЗАШИФРОВАННЫЙ ИЗВЛЕЧЕННЫЙ КЛЮЧ (eDK) 7410
ДЕШИФРОВАНИЕ
7428
ИЗВЛЕЧЕННЫЙ КЛЮЧ (DK)
7412
КОНЕЦ
7440

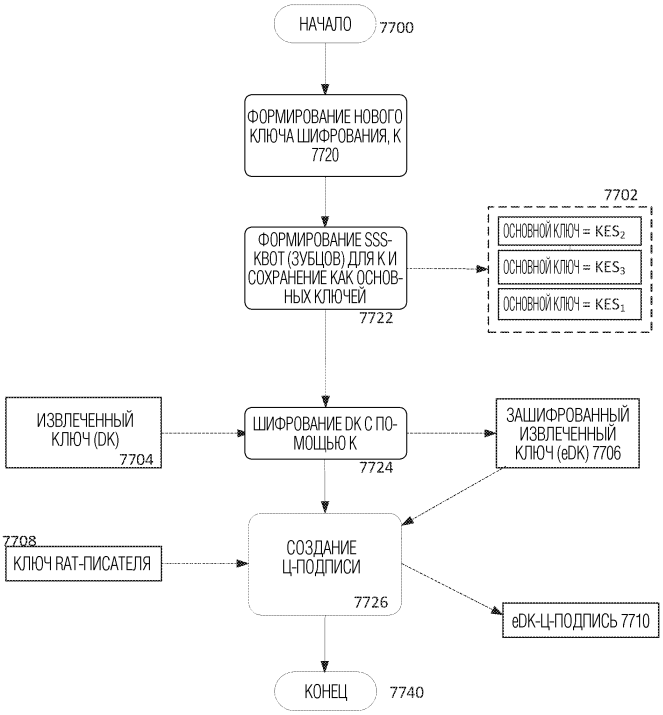
Фиг. 74


НАЧАЛО
7500
7502
ОСНОВНОЙ КЛЮЧ = KES2
ОСНОВНОЙ КЛЮЧ = KES3
ОСНОВНОЙ КЛЮЧ = KES1
СОРТИРОВКА СИММЕ-ТРИЧНЫХ ОСНОВНЫХ КЛЮЧЕЙ В УБЫВАЮЩЕМ ПОРЯДКЕ
7520
ОСНОВНОЙ КЛЮЧ= KES1
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
7504
КОНКАТЕНАЦИЯ ОСНОВ-НЫХ КЛЮЧЕЙ В ОТСОРТИ-РОВАННОМ ПОРЯДКЕ 7522
HASH (SHA 256)
7524
ИЗВЛЕЧЕННЫЙ КЛЮЧ (DK)
7506
ШИФРОВАНИЕ
7526
ЗАШИФРОВАННЫЙ ИЗВЛЕЧЕННЫЙ КЛЮЧ (eDK) 7510
7508
КЛЮЧ RAT-ПИСАТЕЛЯ
СОЗДАНИЕ Ц-ПОДПИСИ
7528
eDK-Ц-ПОДПИСЬ 7512
КОНЕЦ
7540

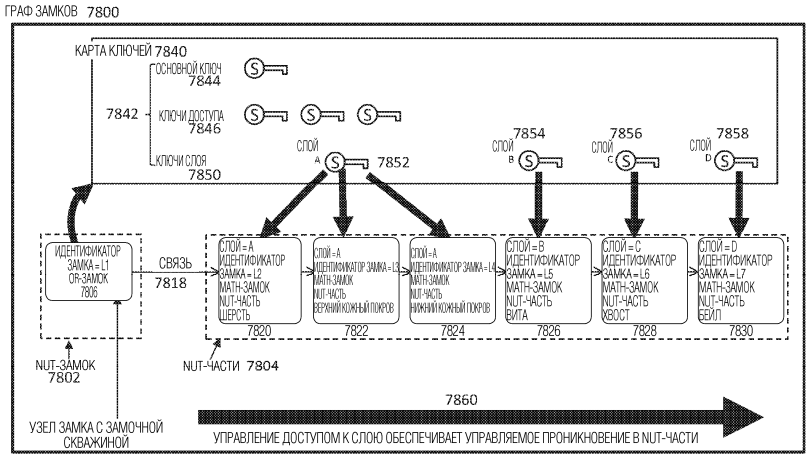
Фиг. 75


НАЧАЛО
7600
7602
КЛЮЧ RAT-ПИСАТЕЛЯ
eDK-Ц-ПОДПИСЬ
7604
АУТЕНТИФИКАЦИЯ Ц-ПОДПИСИ EDK С ИСПОЛЬЗОВАНИЕМ КЛЮЧА RAT-ЧИТАТЕЛЯ УСПЕШНО?
7620
НЕТ
ОШИБКА
7630
ДА
7606
ОСНОВНОЙ КЛЮЧ= KES2
ОСНОВНОЙ КЛЮЧ= KES3
ОСНОВНОЙ КЛЮЧ= KES1
ВОССТАНОВЛЕНИЕ SSS-СЕКРЕТА ИЗ ЗУБЦОВ 7622
ЗАШИФРОВАННЫЙ ИЗВЛЕЧЕННЫЙ КЛЮЧ (eDK) 7608
ДЕШИФРОВАНИЕ
7624
ИЗВЛЕЧЕННЫЙ КЛЮЧ (DK)
7610
КОНЕЦ
7640

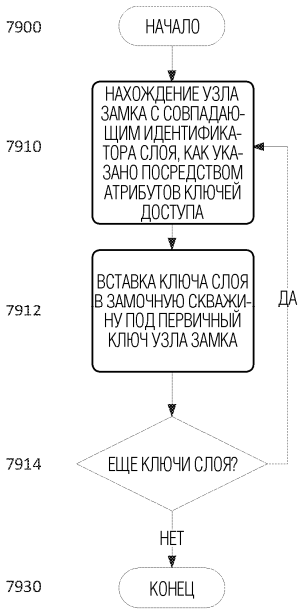
Фиг. 76


НАЧАЛО
7700
ФОРМИРОВАНИЕ НОВОГО КЛЮЧА ШИФРОВАНИЯ, K
7720
7702
ФОРМИРОВАНИЕ SSS-КВОТ (ЗУБЦОВ) ДЛЯ K И СОХРАНЕНИЕ КАК ОСНОВНЫХ КЛЮЧЕЙ
7722
ОСНОВНОЙ КЛЮЧ = KES2
ОСНОВНОЙ КЛЮЧ = KES3
ОСНОВНОЙ КЛЮЧ = KES1
ИЗВЛЕЧЕННЫЙ КЛЮЧ (DK)
7704
ШИФРОВАНИЕ DK С ПОМОЩЬЮ K
7724
ЗАШИФРОВАННЫЙ ИЗВЛЕЧЕННЫЙ КЛЮЧ (eDK) 7706
7708
КЛЮЧ RAT-ПИСАТЕЛЯ
СОЗДАНИЕ Ц-ПОДПИСИ
7726
eDK-Ц-ПОДПИСЬ 7710
КОНЕЦ
7740

Фиг. 77



Фиг. 78



Фиг. 79

РОЛЬ	ХРАНИМЫЙ КЛЮЧ	ДАННЫЕ		Ц-ПОДПИСЬ (ЦИФРОВАЯ ПОДПИСЬ)	
		ШИФРОВАНИЕ	ДЕШИФРОВАНИЕ	СОЗДАНИЕ	ВЕРИФИКАЦИЯ
ЧИТАТЕЛЬ A	S U	ИСТИНА	ИСТИНА	ЛОЖЬ	ИСТИНА
ЧИТАТЕЛЬ B	S U	ИСТИНА	ИСТИНА	ЛОЖЬ	ИСТИНА
ПИСАТЕЛЬ X	S R U	ИСТИНА	ИСТИНА	ИСТИНА	ИСТИНА
ПИСАТЕЛЬ Y	S R U	ИСТИНА	ИСТИНА	ИСТИНА	ИСТИНА
ВЕРИФИКАТОР V	U	ЛОЖЬ	ЛОЖЬ	ЛОЖЬ	ИСТИНА

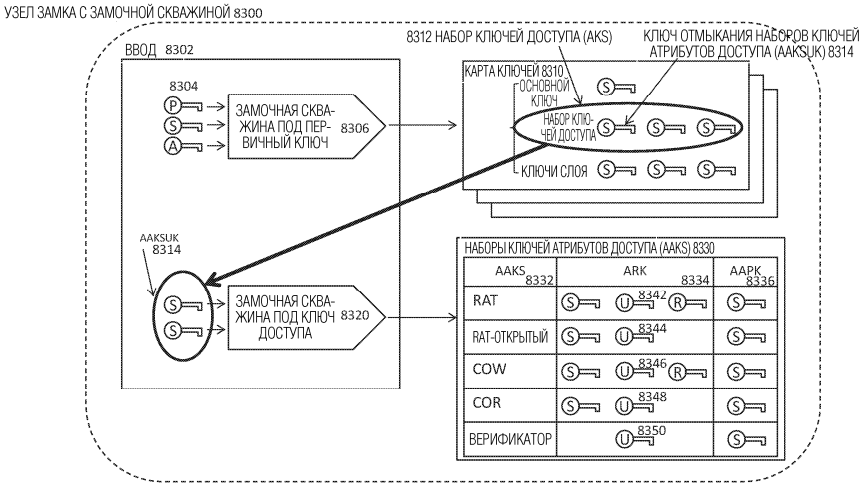
Фиг. 80

ЧАСТЬ	ТИП ДАННЫХ	НЕПРОЗРАЧНОСТЬ МЕШКА	ОПИСАНИЕ
ШЕРСТЬ	МЕТАДАННЫЕ	ПРОЗРАЧНЫЙ	ХАРАКТЕРИСТИКИ ФАЙЛА NUT-КОНТЕЙНЕРА: ИМЯ ФАЙЛА, ИМЯ ПУТИ, РАЗМЕР, ЖУРНАЛ РЕГИСТРАЦИИ ОЧИСТКИ, КАТАЛОГ, ПРАВИЛА ИМЕНОВАНИЯ ФАЙЛОВ/КАТАЛОГОВ
ВЕРХНИЙ КОЖНЫЙ ПОКРОВ	МЕТАДАННЫЕ	ПРОЗРАЧНЫЙ	NUT ХАРАКТЕРИСТИКИ: NUT-ИДЕНТИФИКАТОР, ВЕРСИЯ, ИСТЕЧЕНИЕ СРОКА ДЕЙСТВИЯ, ВРЕМЕННЫЕ МЕТКИ И Т.Д.
НИЖНИЙ КОЖНЫЙ ПОКРОВ	МЕТАДАННЫЕ	ПРОЗРАЧНЫЙ	NUT-ФУНКЦИИ: ВЕРСИИ КРИПТОГРАФИЧЕСКОЙ БИБЛИОТЕКИ, ВЕРСИИ ШИФРА, ХЭШ-ВЕРСИИ
ВИТА	ЖУРНАЛ РЕГИСТРАЦИИ	ЗАШИФРОВАННЫЙ	NUT-РЕГИСТРАЦИЯ
МОРДА	ФАСАД	ПРОЗРАЧНЫЙ	ОТФИЛЬТРОВАННАЯ СВОДКА НА ОСНОВЕ ПРАВИЛ РАБОЧИХ ДАННЫХ
ХВОСТ	ПРЕДЫСТОРИЯ	ЗАШИФРОВАННЫЙ	ПРЕДЫСТОРИЯ ИЗМЕНЕНИЙ РАБОЧИХ ДАННЫХ
БЕЙЛ	РАБОЧИЕ ДАННЫЕ	ЗАШИФРОВАННЫЙ	ОСНОВНЫЕ СОХРАНЕННЫЕ ДАННЫЕ

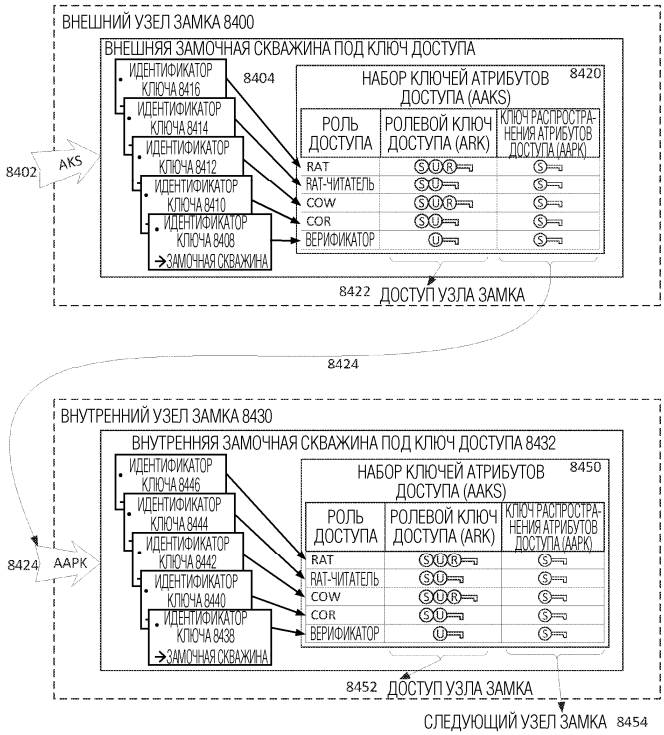
Фиг. 81

NUT-ЧАСТЬ	РОЛЬ ДЛЯ ДОСТУПА	ОПИСАНИЕ
ВСЕ	RAT	NUT-ВЛАДЕЛЕЦ
	RAT-ЧИТАТЕЛЬ	ВЕРИФИКАЦИЯ/СЧИТЫВАНИЕ Ц-ПОДПИСЕЙ NUT-КОНТЕЙНЕРА
	RAT-ВЕРИФИКАТОР	ВЕРИФИКАЦИЯ
БЕЙЛ	COW	ПИСАТЕЛЬ РАБОЧИХ ДАННЫХ (КЛАСС ПИСАТЕЛЕЙ)
	COR	ВЕРИФИКАЦИЯ/СЧИТЫВАНИЕ РАБОЧИХ ДАННЫХ (КЛАСС ЧИТАТЕЛЕЙ)
	COR-ВЕРИФИКАТОР	ВЕРИФИКАЦИЯ
ВИТА	РЕГИСТРАТОР	ПИСАТЕЛЬ ЖУРНАЛА РЕГИСТРАЦИИ
	ЧИТАТЕЛЬ ЖУРНАЛА РЕГИСТРАЦИИ	ВЕРИФИКАЦИЯ/СЧИТЫВАНИЕ ЖУРНАЛА РЕГИСТРАЦИИ
	ВЕРИФИКАТОР ЖУРНАЛА РЕГИСТРАЦИИ	ВЕРИФИКАЦИЯ
ХВОСТ	СЕРВЕР АРХИВНЫХ ДАННЫХ	ПИСАТЕЛЬ ПРЕДЫСТОРИИ
	ЧИТАТЕЛЬ ПРЕДЫСТОРИИ	ВЕРИФИКАЦИЯ/СЧИТЫВАНИЕ ПРЕДЫСТОРИИ
	ВЕРИФИКАТОР ПРЕДЫСТОРИИ	ВЕРИФИКАЦИЯ

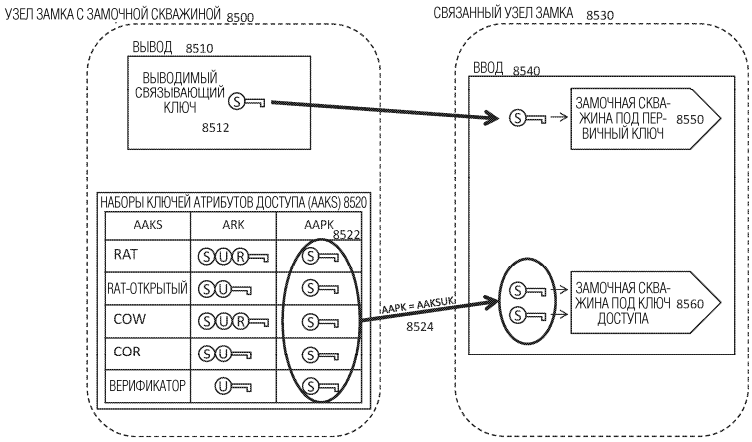
Фиг. 82



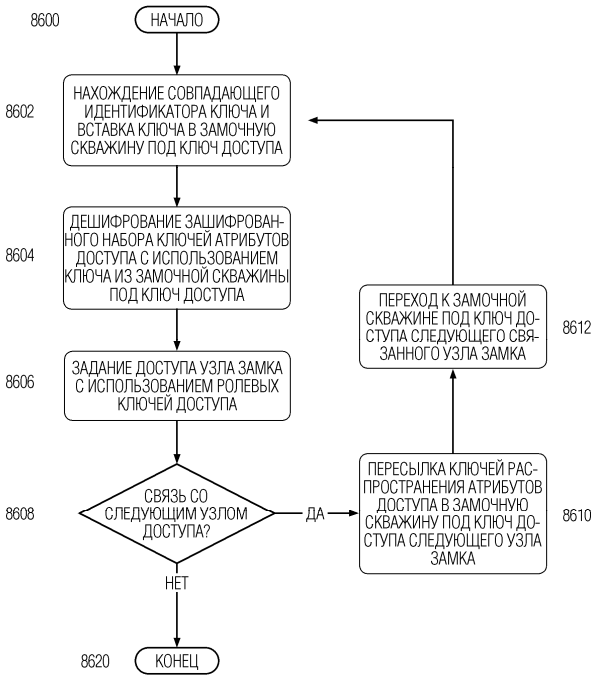
Фиг. 83


ВНЕШНИЙ УЗЕЛ ЗАМКА 8400
ВНЕШНЯЯ ЗАМОЧНАЯ СКВАЖИНА ПОД КЛЮЧ ДОСТУПА
ИДЕНТИФИКАТОР КЛЮЧА 8416
ИДЕНТИФИКАТОР КЛЮЧА 8414
ИДЕНТИФИКАТОР КЛЮЧА 8412
ИДЕНТИФИКАТОР КЛЮЧА 8410
ИДЕНТИФИКАТОР КЛЮЧА 8408
→ЗАМОЧНАЯ СКВАЖИНА
8404
НАБОР КЛЮЧЕЙ АТРИБУТОВ ДОСТУПА (AAKS)
8420
РОЛЬ ДОСТУПА
РОЛЕВОЙ КЛЮЧ ДОСТУПА (ARK)
КЛЮЧ РАСПРОСТРАНЕНИЯ АТРИБУТОВ ДОСТУПА (AAPK)
RAT
RAT-ЧИТАТЕЛЬ
COW
COR
ВЕРИФИКАТОР
8402
AKS
8422 ДОСТУП УЗЛА ЗАМКА
8424
ВНУТРЕННИЙ УЗЕЛ ЗАМКА 8430
ВНУТРЕННЯЯ ЗАМОЧНАЯ СКВАЖИНА ПОД КЛЮЧ ДОСТУПА 8432
ИДЕНТИФИКАТОР КЛЮЧА 8446
ИДЕНТИФИКАТОР КЛЮЧА 8444
ИДЕНТИФИКАТОР КЛЮЧА 8442
ИДЕНТИФИКАТОР КЛЮЧА 8440
ИДЕНТИФИКАТОР КЛЮЧА 8438
→ЗАМОЧНАЯ СКВАЖИНА
НАБОР КЛЮЧЕЙ АТРИБУТОВ ДОСТУПА (AAKS)
8450
РОЛЬ ДОСТУПА
РОЛЕВОЙ КЛЮЧ ДОСТУПА (ARK)
КЛЮЧ РАСПРОСТРАНЕНИЯ АТРИБУТОВ ДОСТУПА (AAPK)
RAT
RAT-ЧИТАТЕЛЬ
COW
COR
ВЕРИФИКАТОР
8424
AAPK
8452 ДОСТУП УЗЛА ЗАМКА
СЛЕДУЮЩИЙ УЗЕЛ ЗАМКА 8454

Фиг. 84


УЗЕЛ ЗАМКА С ЗАМОЧНОЙ СКВАЖИНОЙ 8500
ВЫВОД 8510
ВЫВОДИМЫЙ СВЯЗЫВАЮЩИЙ КЛЮЧ
8512
НАБОРЫ КЛЮЧЕЙ АТРИБУТОВ ДОСТУПА (AAKS) 8520
AAKS
ARK
AAPK
8522
RAT
RAT-ОТКРЫТЫЙ
COW
COR
ВЕРИФИКАТОР
AAPK = AAKSUK
8524
СВЯЗАННЫЙ УЗЕЛ ЗАМКА 8530
ВВОД 8540
ЗАМОЧНАЯ СКВАЖИНА ПОД ПЕРВИЧНЫЙ КЛЮЧ 8550
ЗАМОЧНАЯ СКВАЖИНА ПОД КЛЮЧ ДОСТУПА 8560

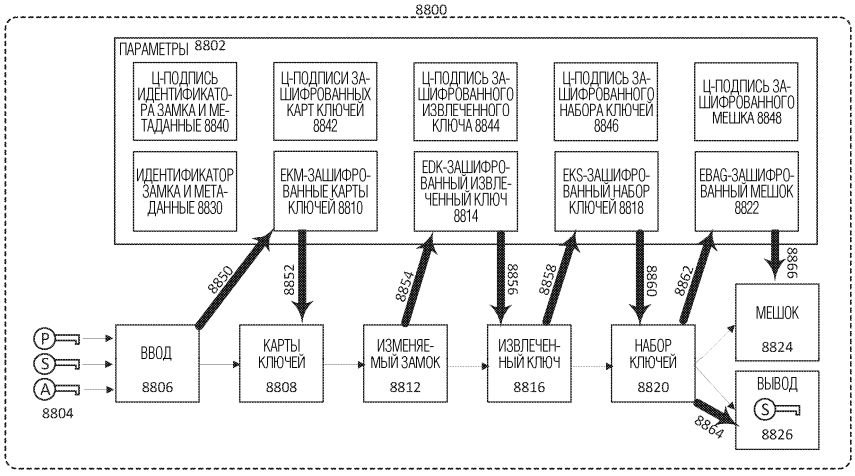
Фиг. 85



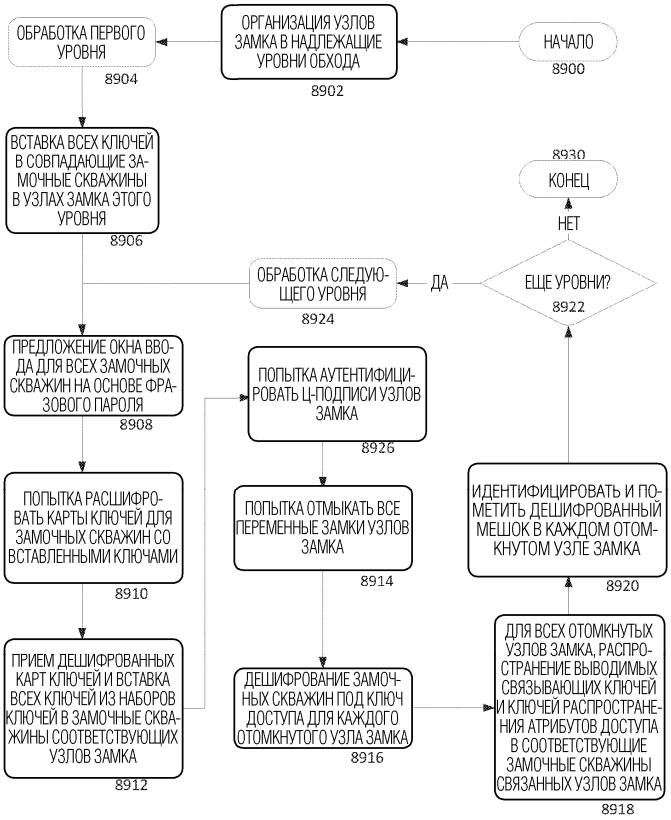
Фиг. 86

РОЛЬ	Ц-ПОДПИСЬ		ЗАПИСЬ		ДАННЫЕ (ПО УМОЛЧАНИЮ)	ДАННЫЕ (ЭКЗЕМПЛЯР)
	СОЗДАНИЕ	ВЕРИФИКАЦИЯ	ШИФРОВАНИЕ	ДЕШИФРОВАНИЕ		
ЧИТАТЕЛЬ		U_D		R_W	S_0	S_n
ПИСАТЕЛЬ	R_D	U_D	U_W	R_W	S_0	S_n
ВЕРИФИКАТОР		U_D				
ТОЛЬКО ДЛЯ ЗАПИСИ	R_D	U_D	U_W			S_n

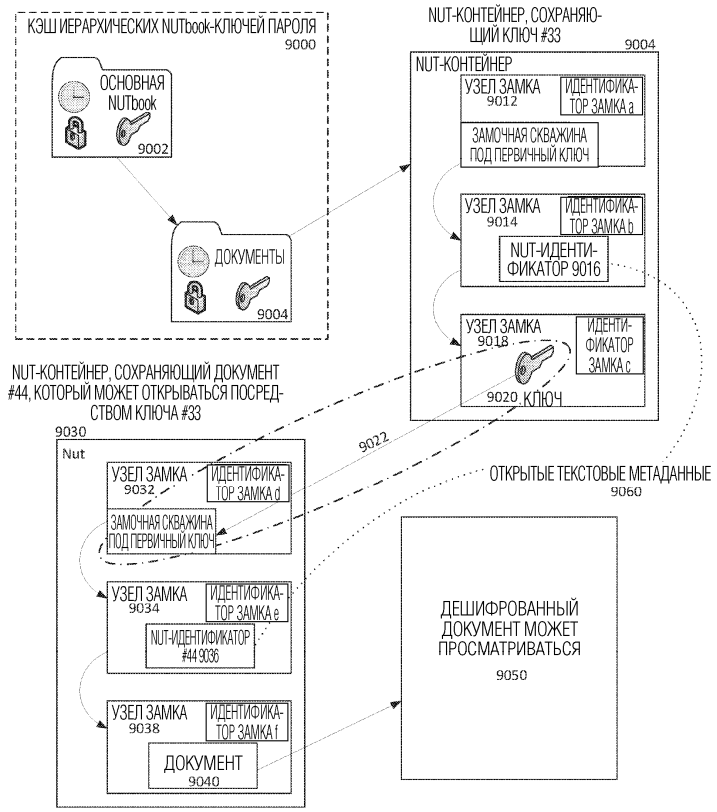
Фиг. 87



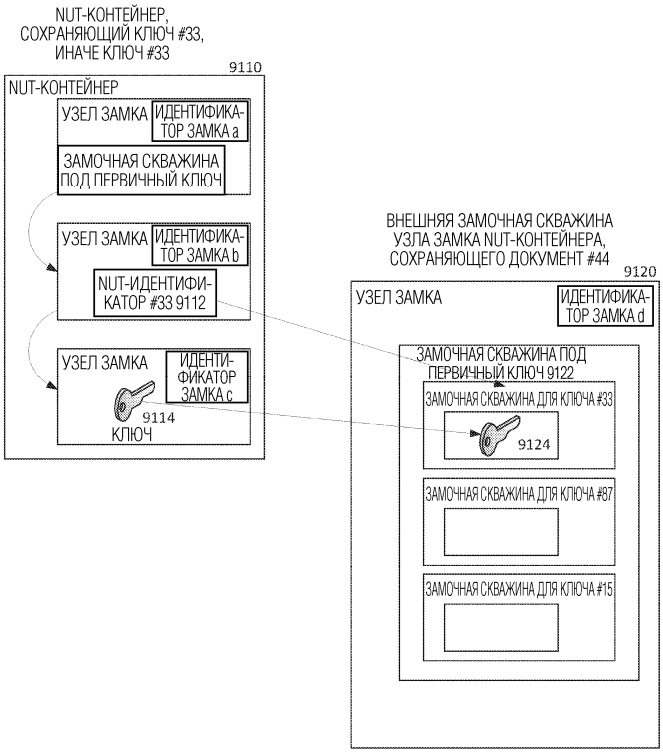
Фиг. 88


НАЧАЛО
8900
ОРГАНИЗАЦИЯ УЗЛОВ ЗАМКА В НАДЛЕЖАЩИЕ УРОВНИ ОБХОДА
8902
ОБРАБОТКА ПЕРВОГО УРОВНЯ
8904
ВСТАВКА ВСЕХ КЛЮЧЕЙ В СОВПАДАЮЩИЕ ЗАМОЧНЫЕ СКВАЖИНЫ В УЗЛАХ ЗАМКА ЭТОГО УРОВНЯ
8906
ПРЕДЛОЖЕНИЕ ОКНА ВВОДА ДЛЯ ВСЕХ ЗАМОЧНЫХ СКВАЖИН НА ОСНОВЕ ФРАЗОВОГО ПАРОЛЯ
8908
ПОПЫТКА РАСШИФРОВАТЬ КАРТЫ КЛЮЧЕЙ ДЛЯ ЗАМОЧНЫХ СКВАЖИН СО ВСТАВЛЕННЫМИ КЛЮЧАМИ
8910
ПРИЕМ ДЕШИФРОВАННЫХ КАРТ КЛЮЧЕЙ И ВСТАВКА ВСЕХ КЛЮЧЕЙ ИЗ НАБОРОВ КЛЮЧЕЙ В ЗАМОЧНЫЕ СКВАЖИНЫ СООТВЕТСТВУЮЩИХ УЗЛОВ ЗАМКА
8912
ПОПЫТКА АУТЕНТИФИЦИРОВАТЬ Ц-ПОДПИСИ УЗЛОВ ЗАМКА
8926
ПОПЫТКА ОТМЫКАТЬ ВСЕ ПЕРЕМЕННЫЕ ЗАМКИ УЗЛОВ ЗАМКА
8914
ДЕШИФРОВАНИЕ ЗАМОЧНЫХ СКВАЖИН ПОД КЛЮЧ ДОСТУПА ДЛЯ КАЖДОГО ОТОМКНУТОГО УЗЛА ЗАМКА
8916
ДЛЯ ВСЕХ ОТОМКНУТЫХ УЗЛОВ ЗАМКА, РАСПРОСТРАНЕНИЕ ВЫВОДИМЫХ СВЯЗЫВАЮЩИХ КЛЮЧЕЙ И КЛЮЧЕЙ РАСПРОСТРАНЕНИЯ АТРИБУТОВ ДОСТУПА В СООТВЕТСТВУЮЩИЕ ЗАМОЧНЫЕ СКВАЖИНЫ СВЯЗАННЫХ УЗЛОВ ЗАМКА
8918
ИДЕНТИФИЦИРОВАТЬ И ПОМЕТИТЬ ДЕШИФРОВАННЫЙ МЕШОК В КАЖДОМ ОТОМКНУТОМ УЗЛЕ ЗАМКА
8920
ЕЩЕ УРОВНИ?
8922
ДА
НЕТ
ОБРАБОТКА СЛЕДУЮЩЕГО УРОВНЯ
8924
КОНЕЦ
8930

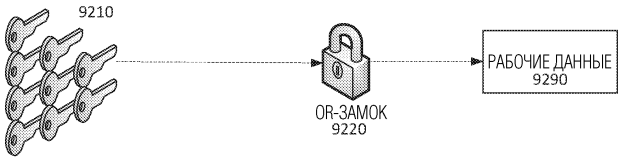
Фиг. 89


КЭШ ИЕРАРХИЧЕСКИХ NUTbook-КЛЮЧЕЙ ПАРОЛЯ
9000
ОСНОВНАЯ NUTbook
9002
ДОКУМЕНТЫ
9004
NUT-КОНТЕЙНЕР, СОХРАНЯЮЩИЙ КЛЮЧ #33
9004
NUT-КОНТЕЙНЕР
УЗЕЛ ЗАМКА 9012
ИДЕНТИФИКАТОР ЗАМКА a
ЗАМОЧНАЯ СКВАЖИНА ПОД ПЕРВИЧНЫЙ КЛЮЧ
УЗЕЛ ЗАМКА 9014
ИДЕНТИФИКАТОР ЗАМКА b
NUT-ИДЕНТИФИКАТОР 9016
УЗЕЛ ЗАМКА 9018
ИДЕНТИФИКАТОР ЗАМКА c
9020 КЛЮЧ
NUT-КОНТЕЙНЕР, СОХРАНЯЮЩИЙ ДОКУМЕНТ #44, КОТОРЫЙ МОЖЕТ ОТКРЫВАТЬСЯ ПОСРЕДСТВОМ КЛЮЧА #33
9030
Nut
9022
УЗЕЛ ЗАМКА 9032
ИДЕНТИФИКАТОР ЗАМКА d
ЗАМОЧНАЯ СКВАЖИНА ПОД ПЕРВИЧНЫЙ КЛЮЧ
УЗЕЛ ЗАМКА 9034
ИДЕНТИФИКАТОР ЗАМКА e
NUT-ИДЕНТИФИКАТОР #44 9036
УЗЕЛ ЗАМКА 9038
ИДЕНТИФИКАТОР ЗАМКА f
ДОКУМЕНТ 9040
ОТКРЫТЫЕ ТЕКСТОВЫЕ МЕТАДАННЫЕ
9060
ДЕШИФРОВАННЫЙ ДОКУМЕНТ МОЖЕТ ПРОСМАТРИВАТЬСЯ
9050

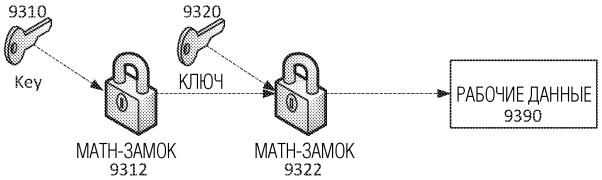
Фиг. 90



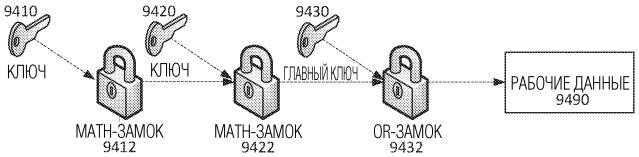
Фиг. 91



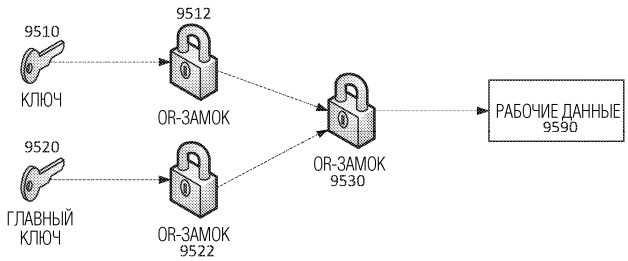
Фиг. 92



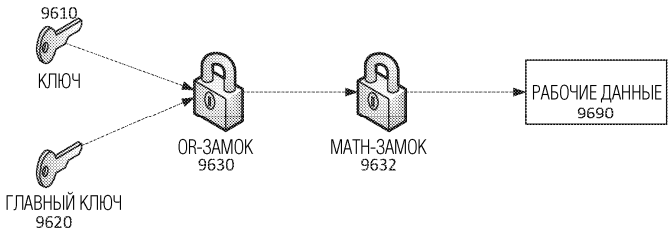
Фиг. 93



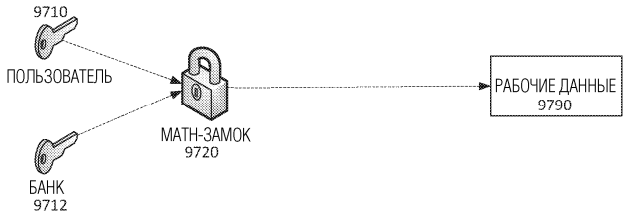
Фиг. 94



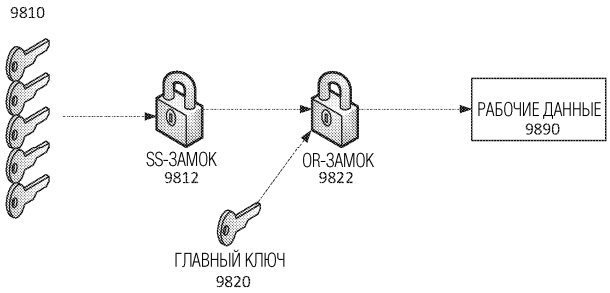
Фиг. 95



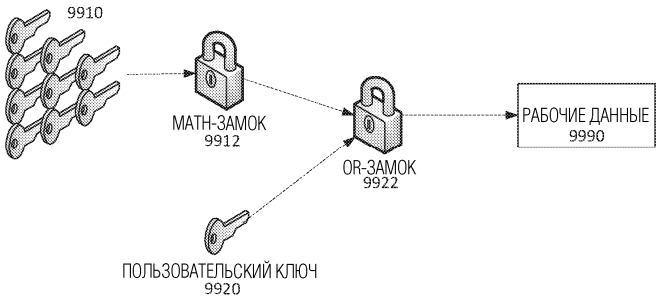
Фиг. 96



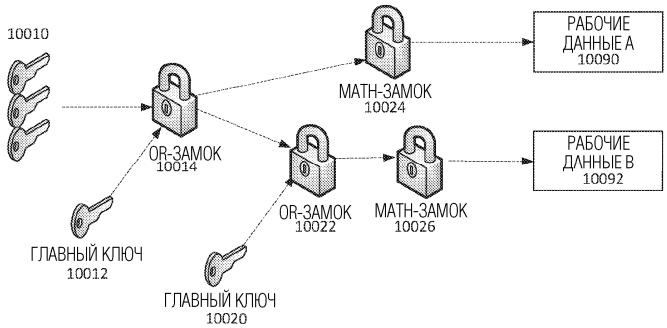
Фиг. 97



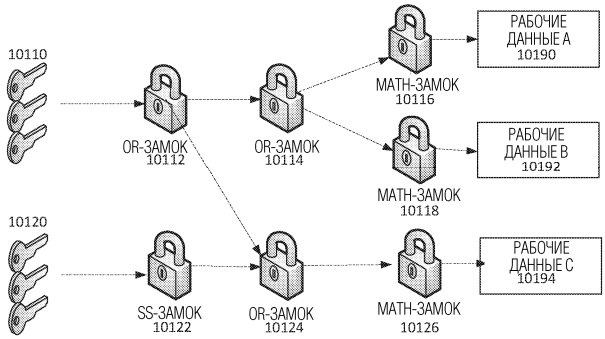
Фиг. 98



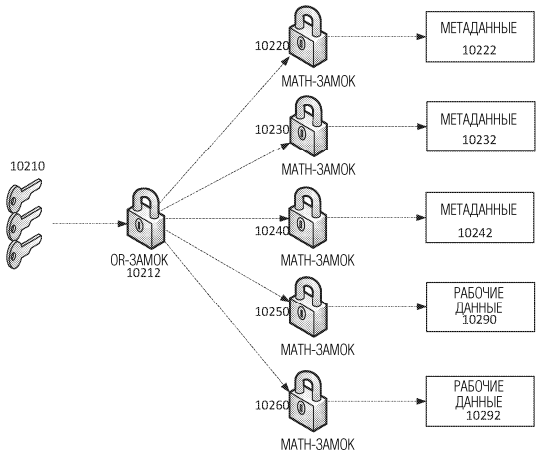
Фиг. 99



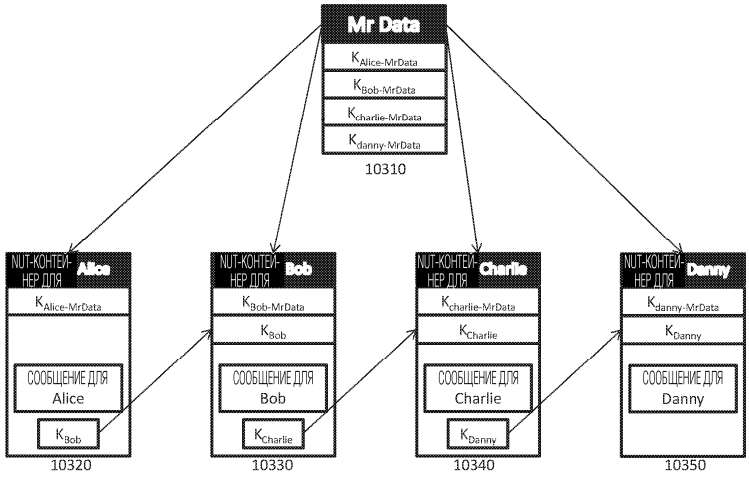
Фиг. 100



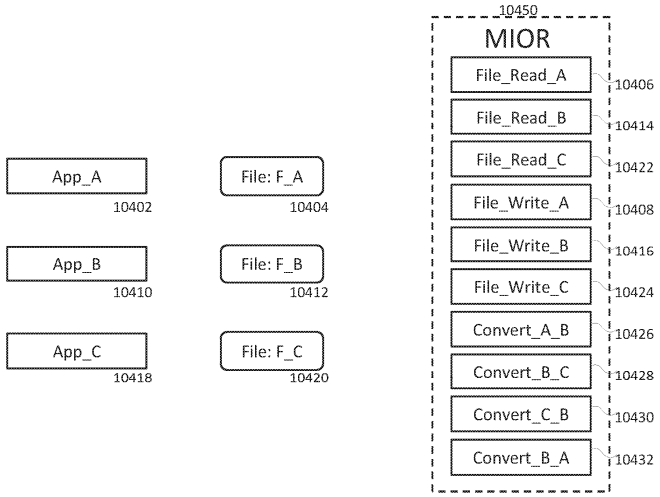
Фиг. 101



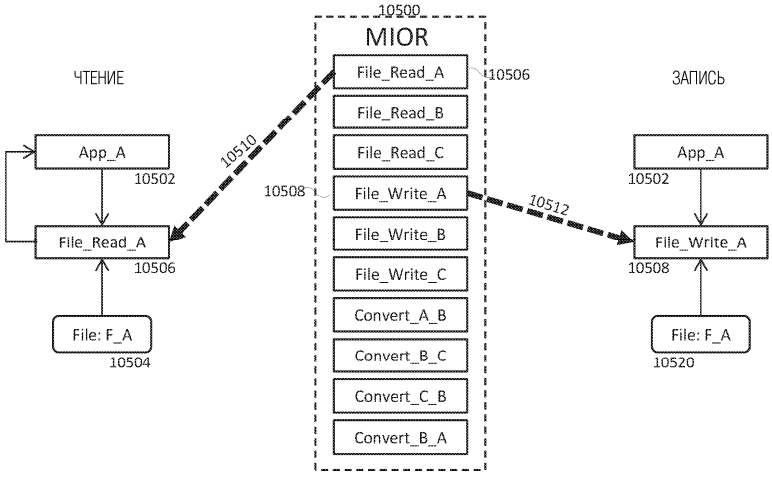
Фиг. 102



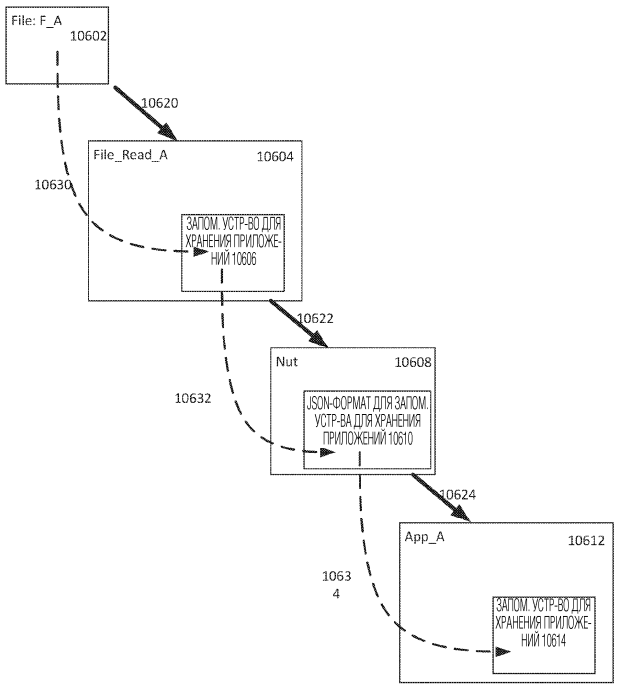
Фиг. 103



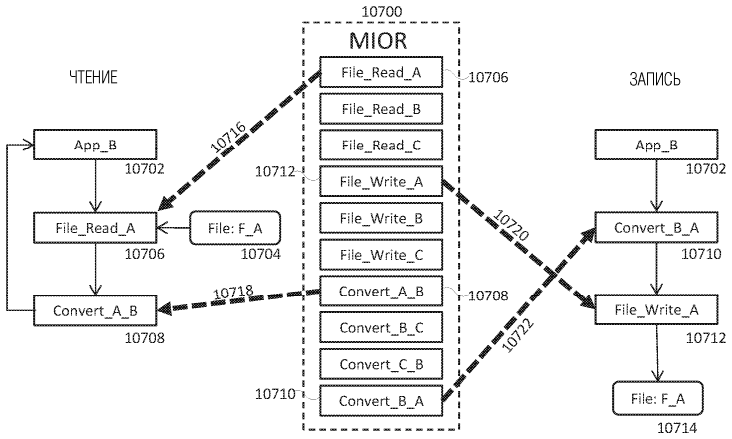
Фиг. 104



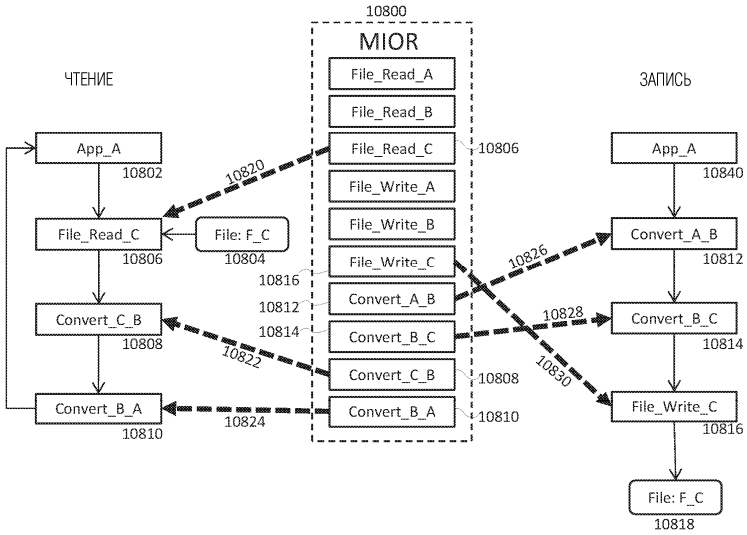
Фиг. 105



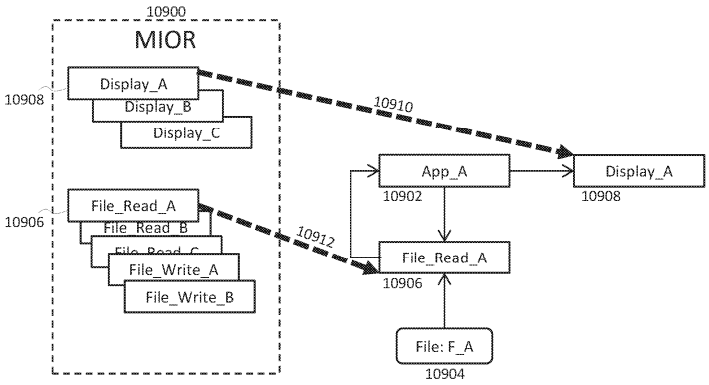
Фиг. 106



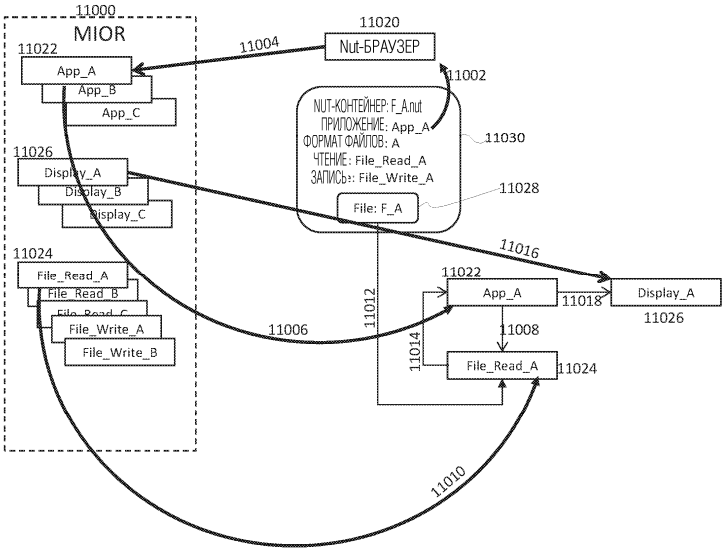
Фиг. 107



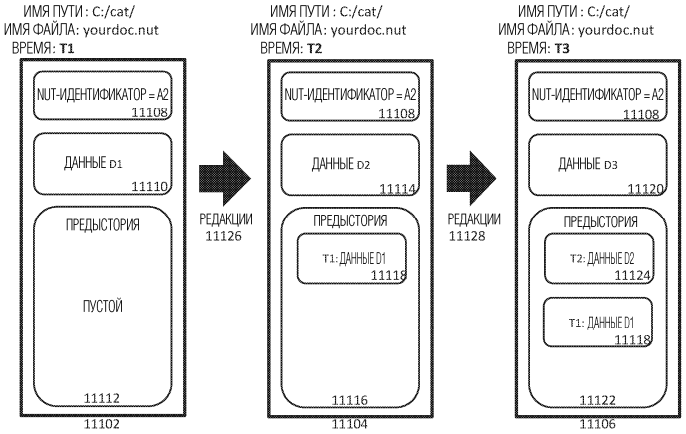
Фиг. 108



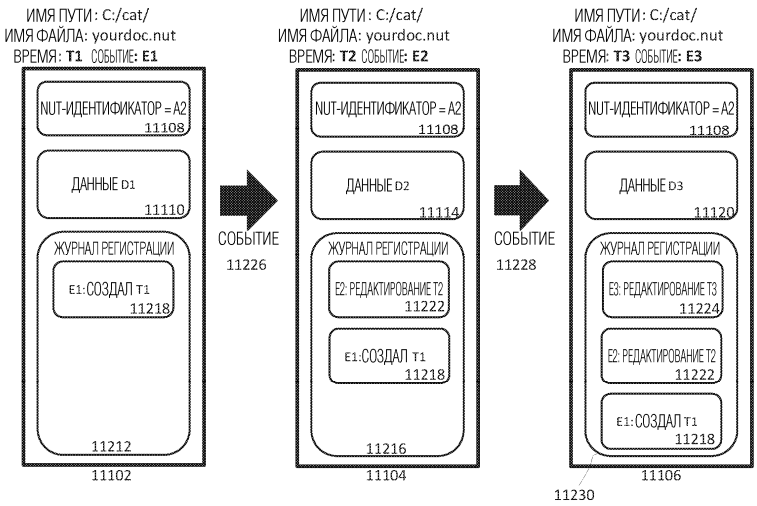
Фиг. 109



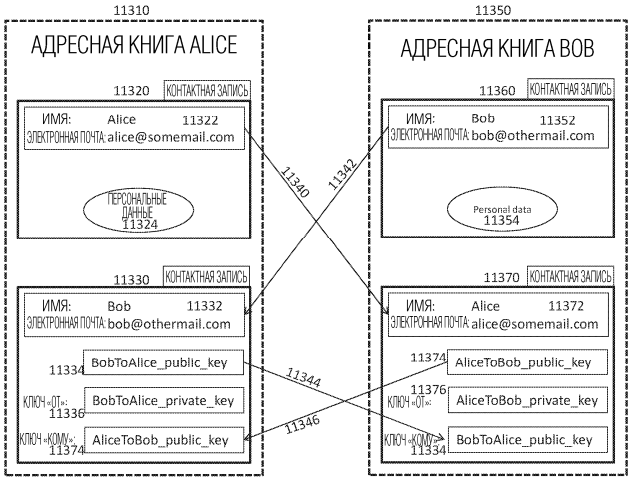
Фиг. 110



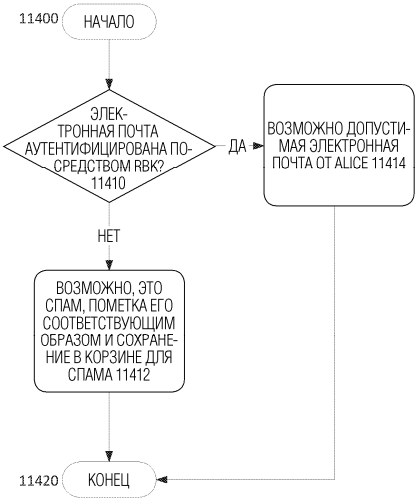
Фиг. 111



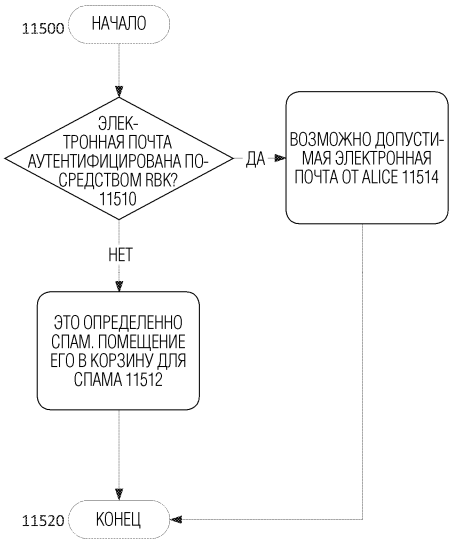
Фиг. 112



Фиг. 113



Фиг. 114



Фиг. 115

СИМПТОМ	ВОЗМОЖНАЯ ПРИЧИНА	ДЕЙСТВИЕ	ЭФФЕКТ
ALICE ПРИНИМАЕТ НЕКОРРЕКТНО ЗАШИФРОВАННЫЙ NUT-КОНТЕЙНЕР ОТ BOB	КТО-ТО ПЫТАЕТСЯ ВЫДАВАТЬ СЕБЯ ЗА BOB	КОНТАКТ С BOB И СБРОС RBK	НОВЫЙ RBK-КАНАЛ УСТАНАВЛИВАЕТСЯ
ALICE ПРИНИМАЕТ СПАМ ОТ BOB	RBK BOB ДЛЯ ALICE СКОМПРОМЕТИРОВАН	КОНТАКТ С BOB И СБРОС RBK	НОВЫЙ RBK-КАНАЛ УСТАНАВЛИВАЕТСЯ
ALICE ПРИНИМАЕТ СПАМ ОТ BOB	BOB ТЕПЕРЬ РАБОТАЕТ В КАЧЕСТВЕ СПАМЕРА	ОДЕРГИВАНИЕ BOB	ПОСМОТРЕТЬ, ПРИСЛУШИВАЕТСЯ ИЛИ НЕТ BOB
ALICE ПРИНИМАЕТ СПАМ ОТ BOB	BOB ТЕПЕРЬ РАБОТАЕТ В КАЧЕСТВЕ СПАМЕРА	СТИРАНИЕ RBK BOB	ПОКА, BOB
ALICE ПРИНИМАЕТ СПАМ ОТ BOB	BOB ПРОДАЛ RBK ДАННЫЕ ALICE СПАМЕРУ	СТИРАНИЕ RBK BOB	ALICE НЕНАВИДИТ BOB

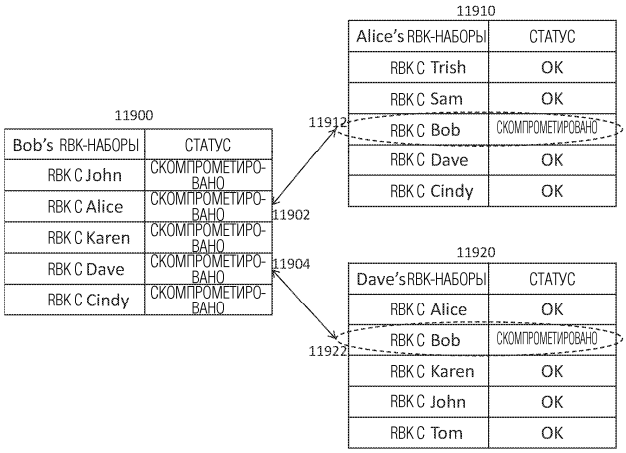
Фиг. 116

СИМПТОМ	ВОЗМОЖНАЯ ПРИЧИНА	ДЕЙСТВИЕ	ЭФФЕКТ
ALICE ПРИНИМАЕТ НЕКОРРЕКТНО ЗАШИФРОВАННЫЙ NUT-КОНТЕЙНЕР ОТ ПРОИЗВОДИТЕЛЯ	КТО-ТО ПЫТАЕТСЯ ВЫДАВАТЬ СЕБЯ ЗА ПРОИЗВОДИТЕЛЯ	КОНТАКТ С ПРОИЗВОДИТЕЛЕМ И СБРОС RBK	НОВЫЙ RBK-КАНАЛ УСТАНАВЛИВАЕТСЯ
ALICE ПРИНИМАЕТ СПАМ ОТ ПРОИЗВОДИТЕЛЯ	RBK ПРОИЗВОДИТЕЛЯ ДЛЯ ALICE СКОМПРОМЕТИРОВАН	КОНТАКТ С ПРОИЗВОДИТЕЛЕМ И СБРОС RBK	НОВЫЙ RBK-КАНАЛ УСТАНАВЛИВАЕТСЯ
ALICE ПРИНИМАЕТ СПАМ ОТ ПРОИЗВОДИТЕЛЯ	ПРОИЗВОДИТЕЛЬ ИМЕЕТ РАЗДРАЖАЮЩИЙ МАРКЕТИНГОВЫЙ ОТДЕЛ	СТИРАНИЕ RBK ПРОИЗВОДИТЕЛЯ	ПОКА, ПРОИЗВОДИТЕЛЬ *
ALICE ПРИНИМАЕТ СПАМ ОТ СПАМЕРА С ИСПОЛЬЗОВАНИЕМ RBK С ПРОИЗВОДИТЕЛЕМ	ПРОИЗВОДИТЕЛЬ ПРОДАЛ ДАННЫЕ ALICE СПАМЕРУ	СТИРАНИЕ RBK ПРОИЗВОДИТЕЛЯ	ALICE НЕНАВИДИТ ПРОИЗВОДИТЕЛЯ
КРЕДИТНАЯ КАРТА ALICE МОШЕННИЧЕСКИ ИСПОЛЬЗУЕТСЯ С RBK С ПРОИЗВОДИТЕЛЕМ	ПРОИЗВОДИТЕЛЬ ВЗЛОМАН	КОНТАКТ С ПРОИЗВОДИТЕЛЕМ, КОМПАНИЕЙ ПО ВЫПУСКУ КРЕДИТНЫХ КАРТ И СБРОС RBK	НОВЫЙ RBK-КАНАЛ УСТАНАВЛИВАЕТСЯ
КРЕДИТНАЯ КАРТА ALICE МОШЕННИЧЕСКИ ИСПОЛЬЗУЕТСЯ С RBK С ПРОИЗВОДИТЕЛЕМ	ПРОИЗВОДИТЕЛЬ ЯВЛЯЕТСЯ МОШЕННИКОМ	КОНТАКТ С ОФИЦИАЛЬНЫМИ ОРГАНАМИ, КОМПАНИЕЙ ПО ВЫПУСКУ КРЕДИТНЫХ КАРТ И СТИРАНИЯ RBK	ПОЗОР ВАМ, ПРОИЗВОДИТЕЛЬ

Фиг. 117

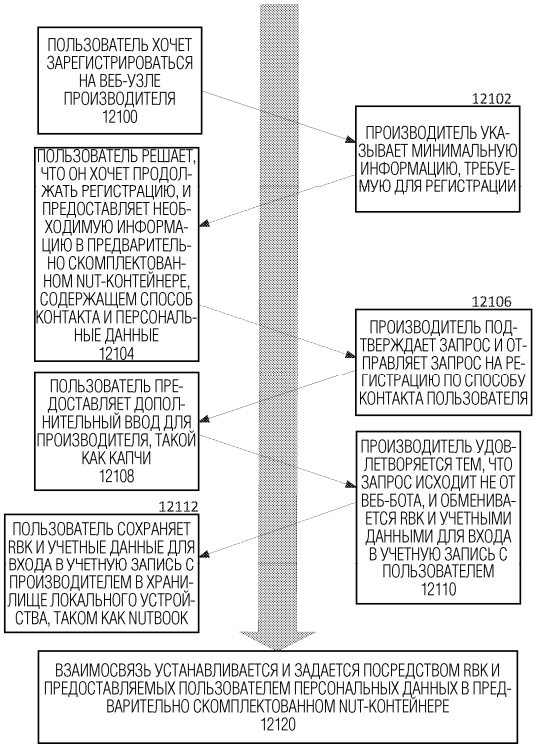
СИМПТОМ	ВОЗМОЖНАЯ ПРИЧИНА	ДЕЙСТВИЕ	ЭФФЕКТ
ПРОИЗВОДИТЕЛЬ ПРИНИМАЕТ НЕКОРРЕКТНО ЗАШИФРОВАННЫЙ NUT-КОНТЕЙНЕР ОТ ALICE	КТО-ТО ПЫТАЕТСЯ ВЫДАВАТЬ СЕБЯ ЗА ALICE	КОНТАКТ С ALICE И СБРОС RBK	НОВЫЙ RBK-КАНАЛ УСТАНАВЛИВАЕТСЯ
ПРОИЗВОДИТЕЛЬ ПРИНИМАЕТ СПАМ ОТ ALICE	RBK ALICE ДЛЯ ПРОИЗВОДИТЕЛЯ СКОМПРОМЕТИРОВАН	КОНТАКТ С ALICE И СБРОС RBK	НОВЫЙ RBK-КАНАЛ УСТАНАВЛИВАЕТСЯ
ПРОИЗВОДИТЕЛЬ ПРИНИМАЕТ СПАМ ОТ ALICE	ALICE ЯВЛЯЕТСЯ СПАМЕРОМ	СТИРАНИЕ RBK ALICE	ПОКА, ALICE *
ПРОИЗВОДИТЕЛЬ ПРИНИМАЕТ СПАМ ОТ СПАМЕРА С ИСПОЛЬЗОВАНИЕМ RBK С ALICE	ALICE ПРОДАЛА ДАННЫЕ ПРОИЗВОДИТЕЛЯ СПАМЕРУ	СТИРАНИЕ RBK ALICE И ПОМЕЩЕНИЕ ALICE ВО ВНУТРЕННИЙ СПИСОК ОСОБОГО КОНТРОЛЯ	ПОКА, ALICE *
ALICE ОТПРАВЛЯЕТ МОШЕННИЧЕСКУЮ ТРАНЗАКЦИЮ	ALICE ИСПОЛЬЗУЕТ УКРАДЕННУЮ КРЕДИТНУЮ КАРТУ	СТИРАНИЕ RBK ALICE И КОНТАКТ С КОМПАНИЕЙ ПО ВЫПУСКУ КРЕДИТНЫХ КАРТ	БЕГ, ALICE, БЕГИ

Фиг. 118

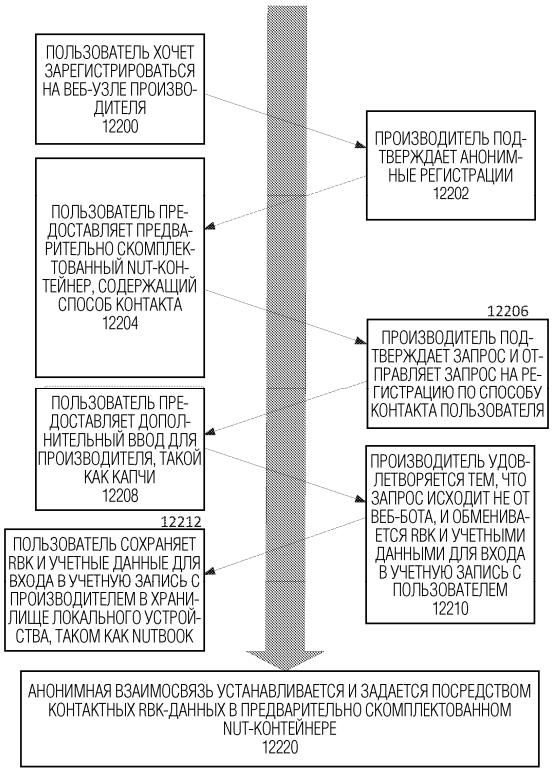


Фиг. 119

Фиг. 120



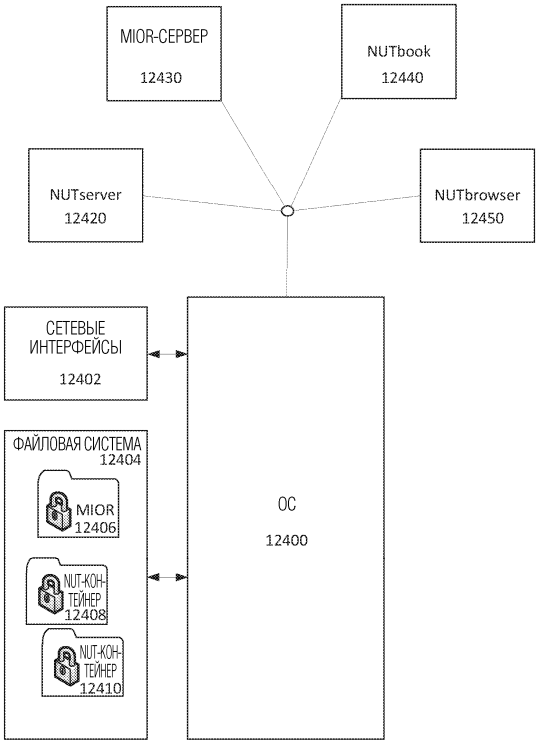
Фиг. 121



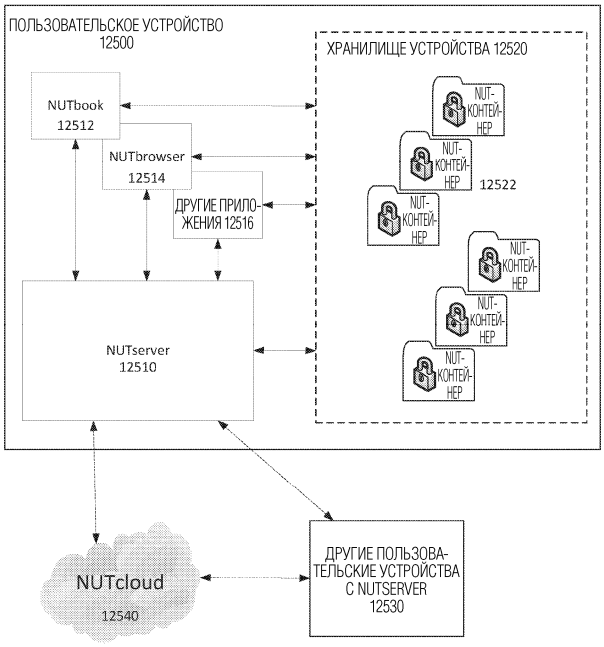
Фиг. 122

УСЛУГА	ОПИСАНИЕ
NUTSERVER	ОРГАНИЗУЕТ ЛОКАЛЬНЫЕ NUT-КОНТЕЙНЕРЫ. СИНХРОНИЗИРУЕТСЯ С РАВНОПРАВНЫМИ УЗЛАМИ. ОСНОВНОЙ ИНТЕРФЕЙС ЛОКАЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ PKI НА ОСНОВЕ NUT-КОНТЕЙНЕРОВ.
MIOR-СЕРВЕР	УПРАВЛЯЕТ ЛОКАЛЬНЫМ MIOR-КЭШЕМ ДЛЯ БЫСТРОГО ИЗВЛЕЧЕНИЯ ЧАСТО ИСПОЛЬЗУЕМЫХ МОДУЛЕЙ. ОБМЕНИВАЕТСЯ ДАННЫМИ С ВНЕШНИМИ MIOR-СЕРВЕРАМИ.
NUTBOOK	ПРИЛОЖЕНИЕ ОРГАНИЗАЦИИ КАТАЛОГОВ НА ОСНОВЕ NUT-КОНТЕЙНЕРОВ. СЛУЖИТ В КАЧЕСТВЕ ОСНОВНОГО PKI НА ОСНОВЕ NUT-КОНТЕЙНЕРОВ.
NUTBROWSER	ОБЩИЙ NUT-БРАУЗЕР, РЕАЛИЗУЮЩИЙ ВСЕ MIOR-ПРИЗНАКИ. ТАКЖЕ ВЫСТУПАЕТ В КАЧЕСТВЕ ФАЙЛОВОГО БРАУЗЕРА.

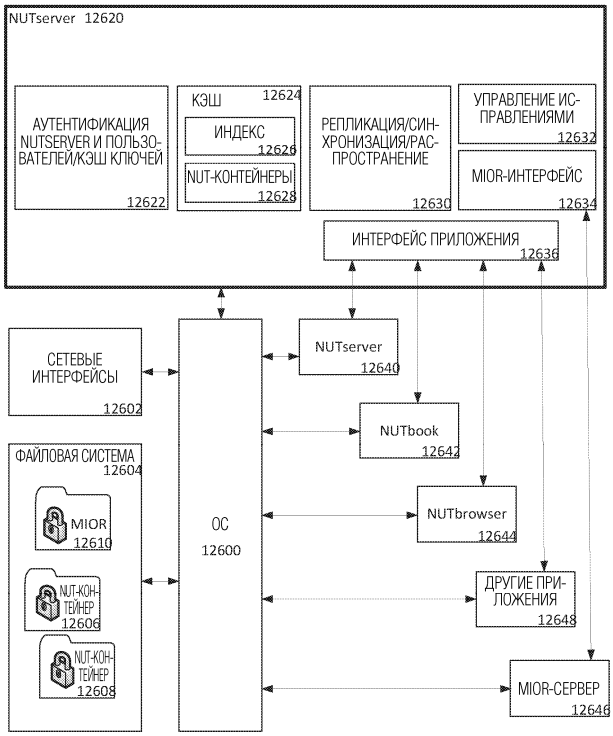
Фиг. 123



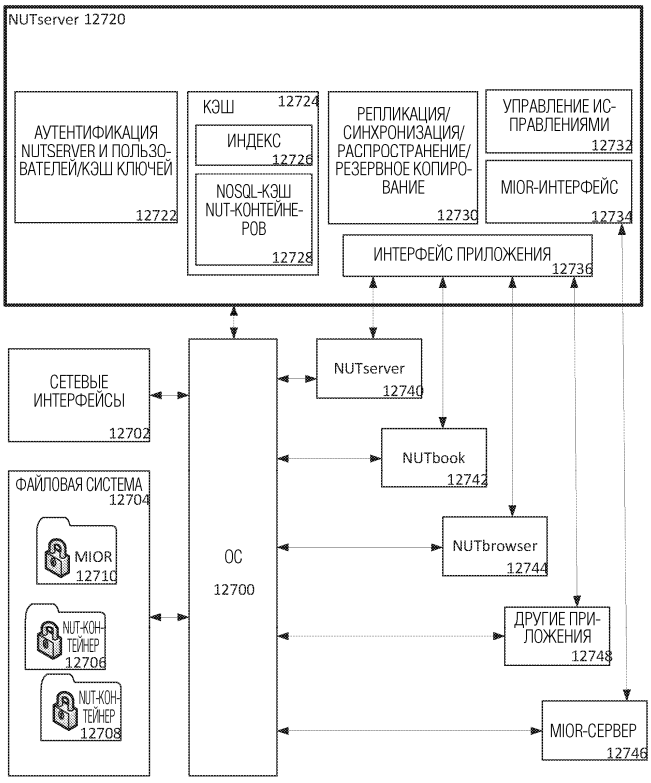
Фиг. 124



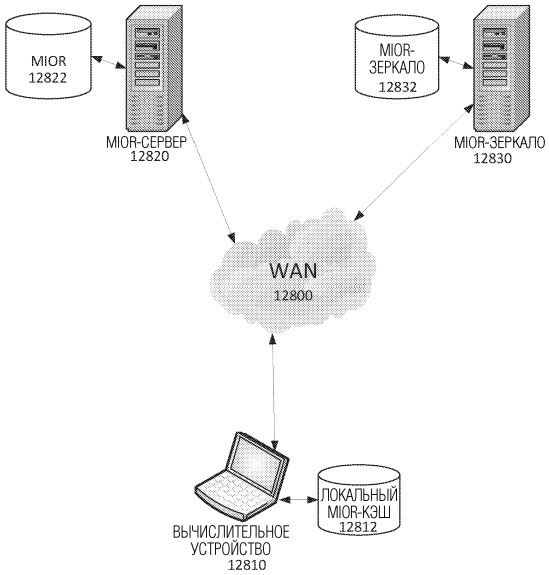
Фиг. 125



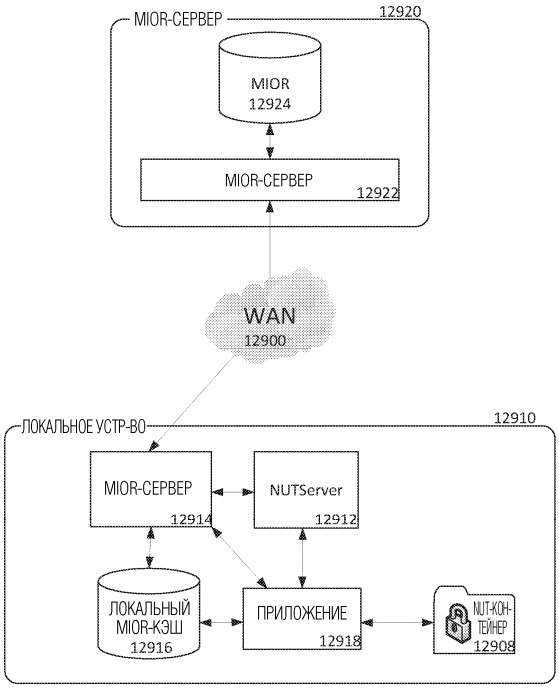
Фиг. 126



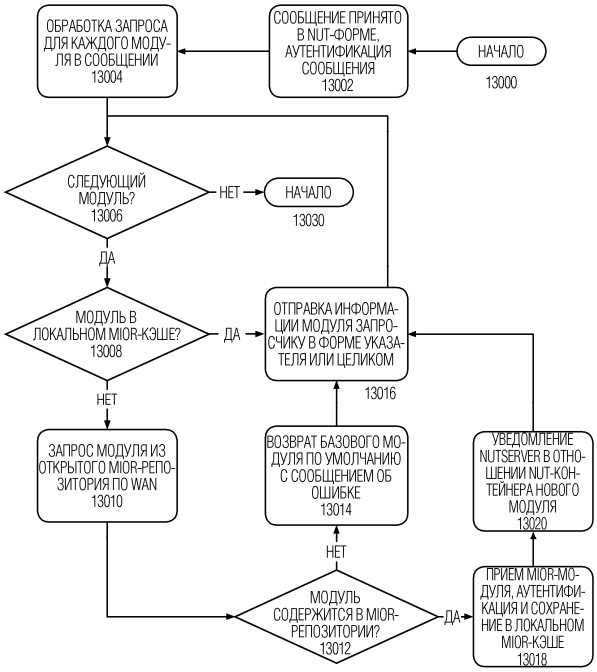
Фиг. 127



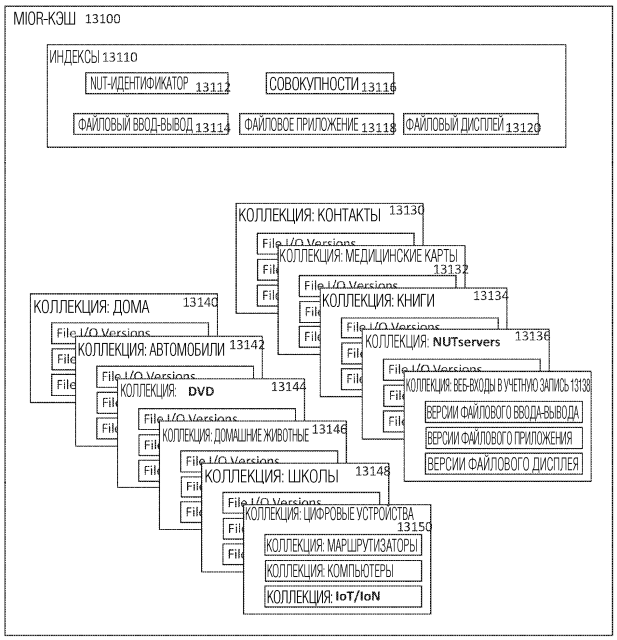
Фиг. 128



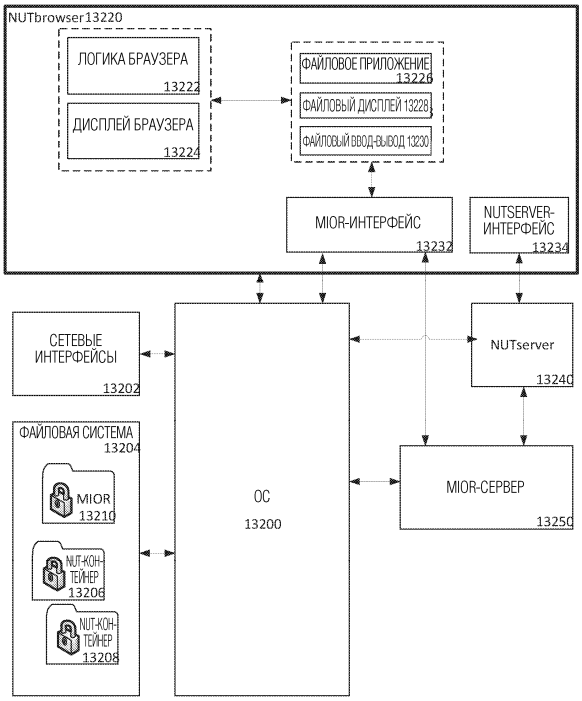
Фиг. 129



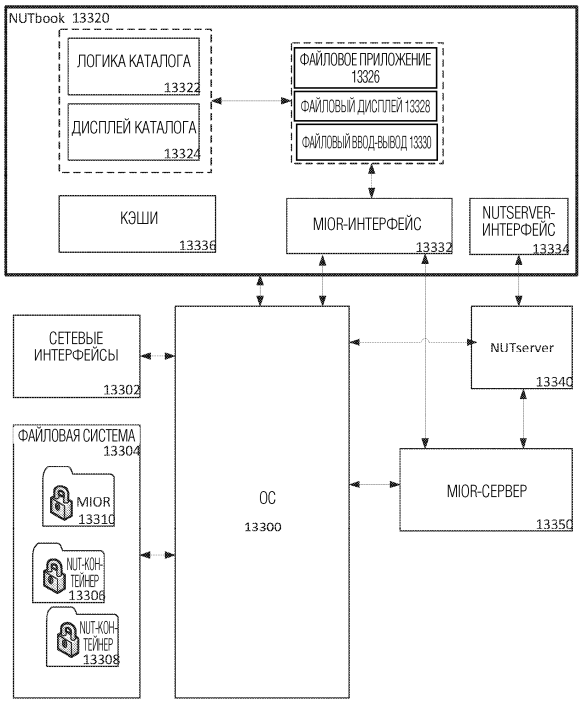
Фиг. 130



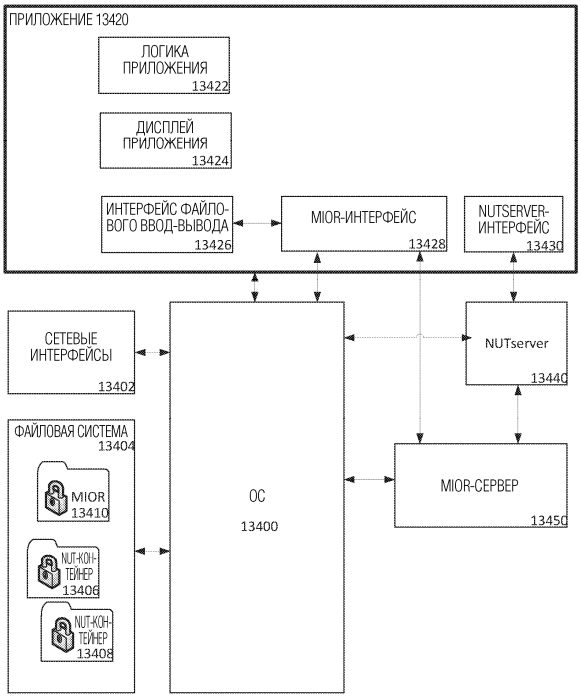
Фиг. 131



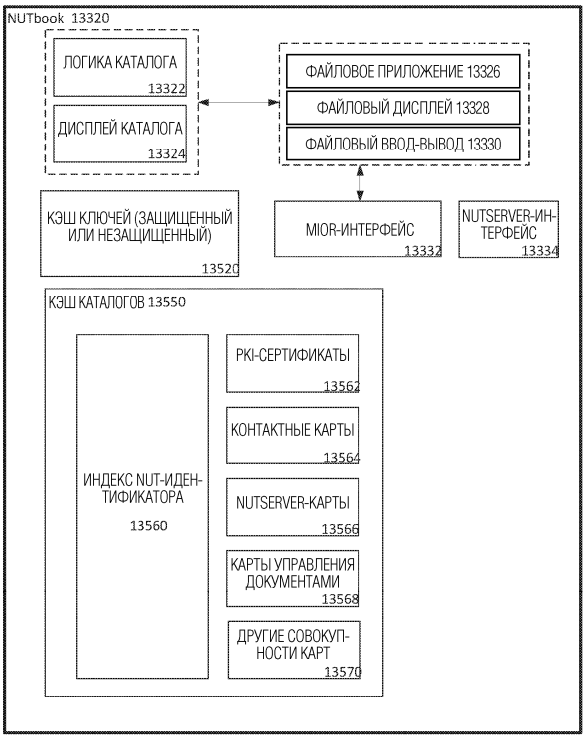
Фиг. 132



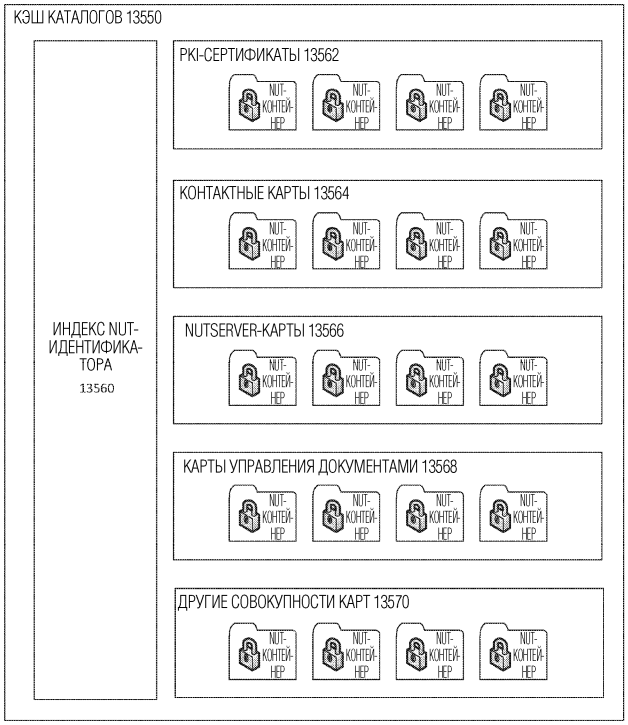
Фиг. 133



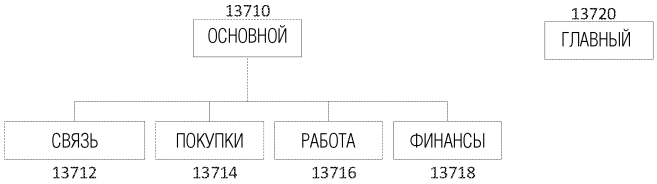
Фиг. 134



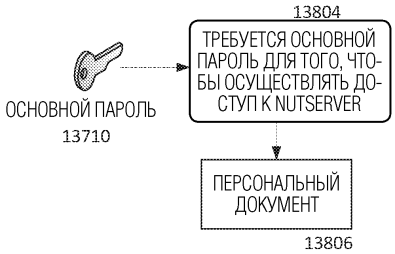
Фиг. 135



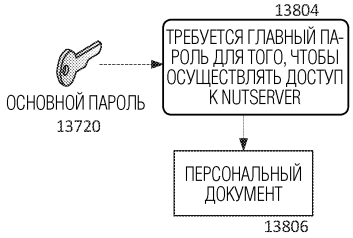
Фиг. 136



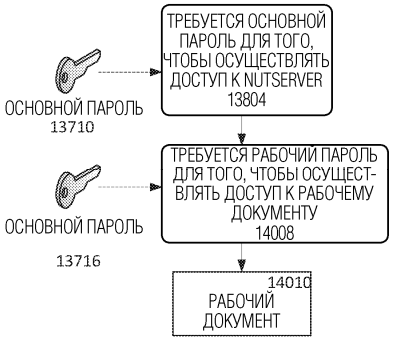
Фиг. 137



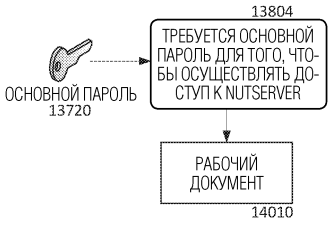
Фиг. 138



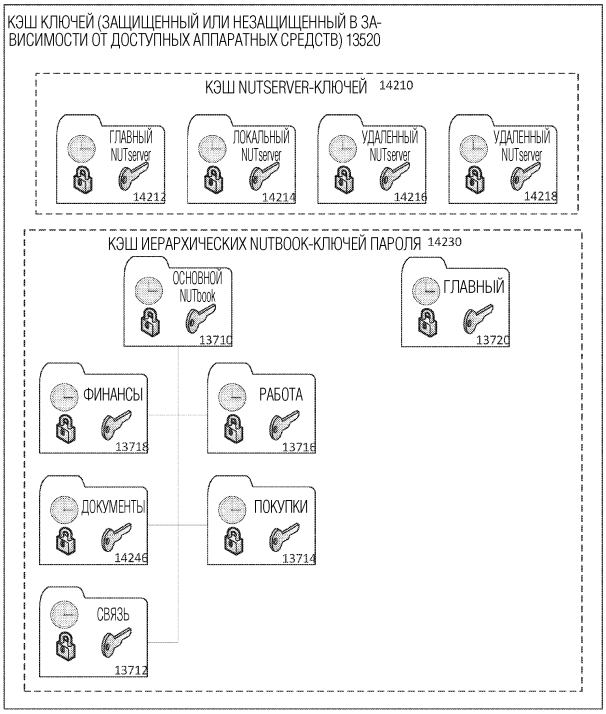
Фиг. 139



Фиг. 140

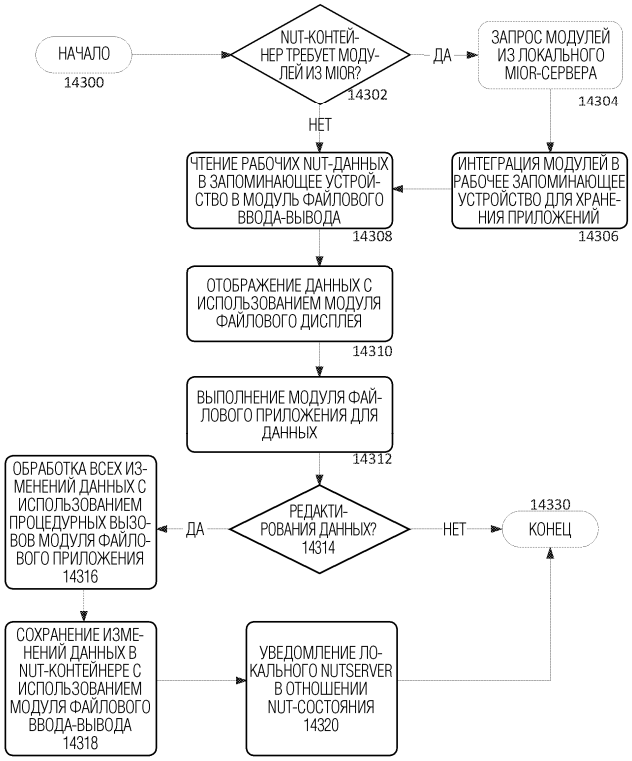


Фиг. 141



Фиг. 142

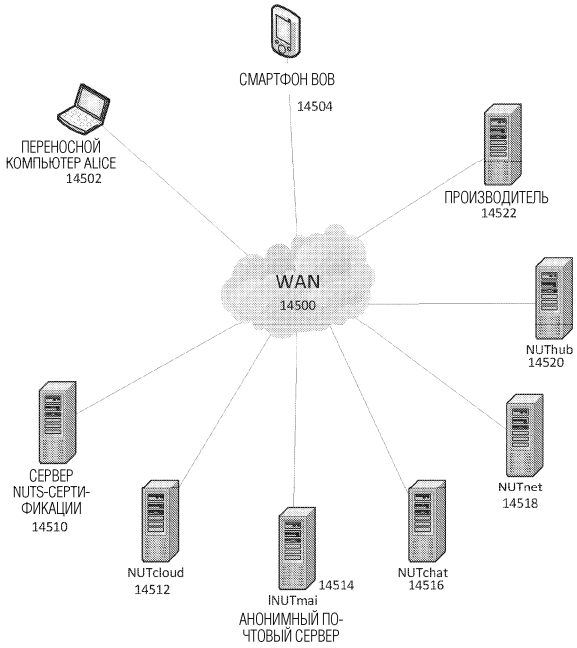
КЭШ ИЕРАРХИЧЕСКИХ NUTBOOK-КЛЮЧЕЙ ПАРОЛЯ



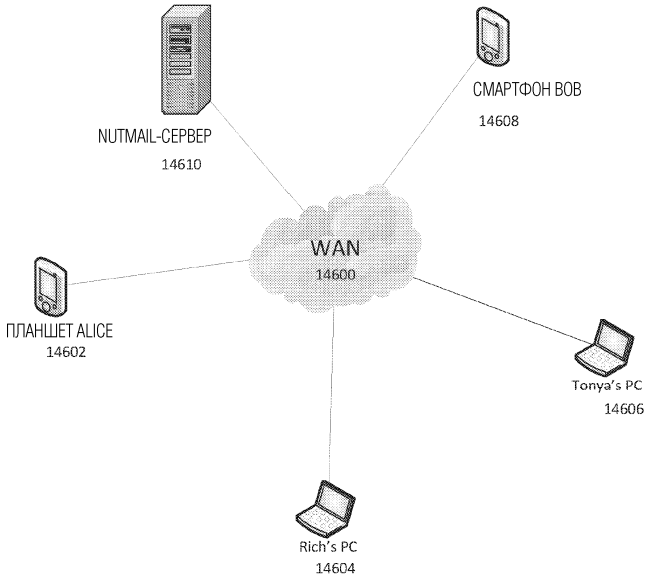
Фиг. 143

УСЛУГА	ОПИСАНИЕ
NUTMAIL	ЗАКРЫТАЯ АНОНИМНАЯ ЭЛЕКТРОННАЯ ПОЧТА
NUTCHAT	ЗАКРЫТЫЙ АНОНИМНЫЙ ЧАТ
NUTCLOUD	ЗАКРЫТОЕ АНОНИМНОЕ ОБЛАЧНОЕ ХРАНИЛИЩЕ ДАННЫХ
NUTNET	ЗАКРЫТАЯ АНОНИМНАЯ СОЦИАЛЬНАЯ СЕТЬ
NUTHUB	ЗАКРЫТАЯ АНОНИМНАЯ ПЕРЕАДРЕСАЦИЯ ПОРТОВ. ПОДДЕРЖИВАЕТ ION (ИНТЕРНЕТ NUTS)
NUTCERTIFICATION	УСЛУГА ПРОВЕРКИ ЦЕЛОСТНОСТИ КОММЕРЧЕСКОГО NUTSERVER

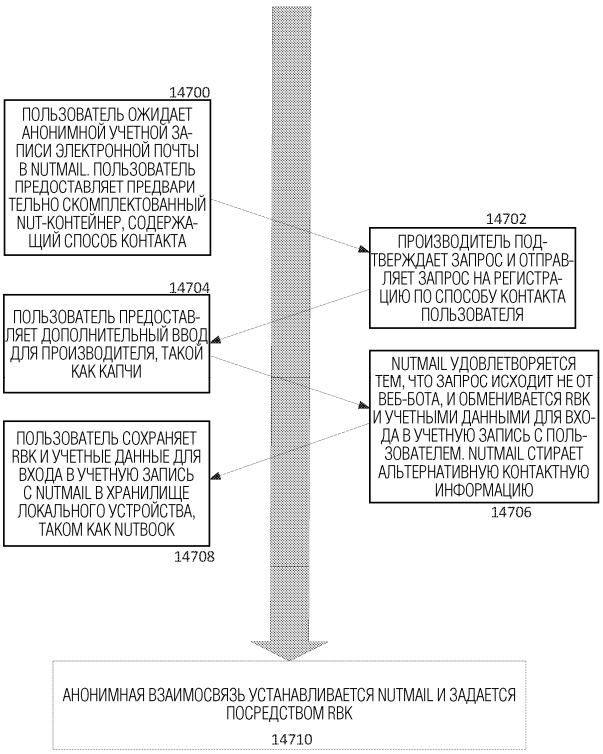
Фиг. 144



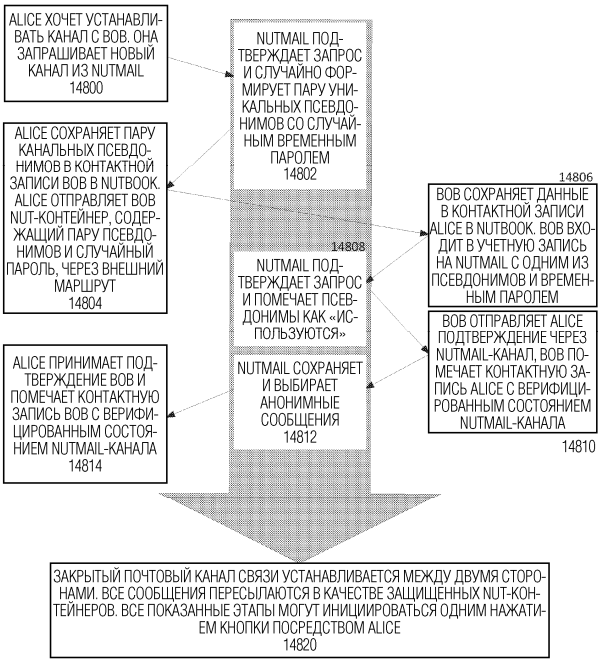
Фиг. 145



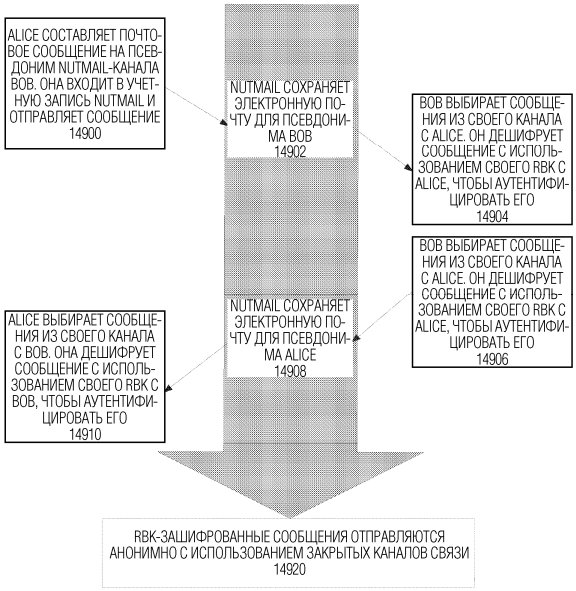
Фиг. 146



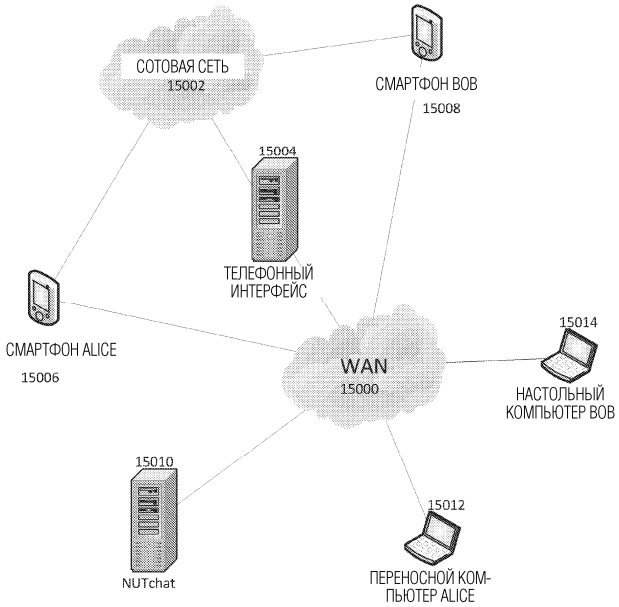
Фиг. 147



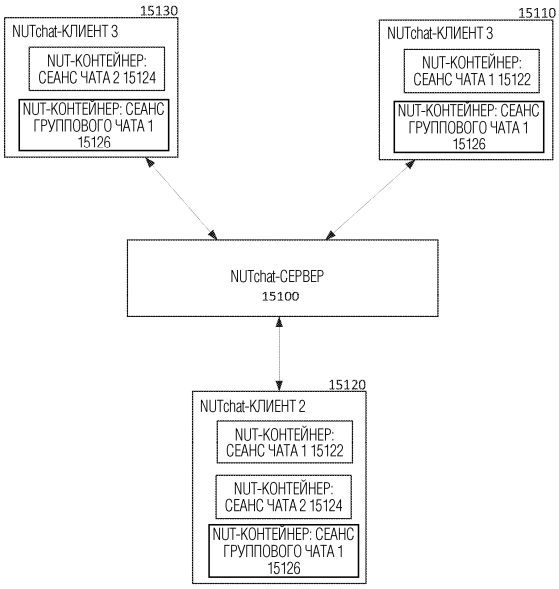
Фиг. 148



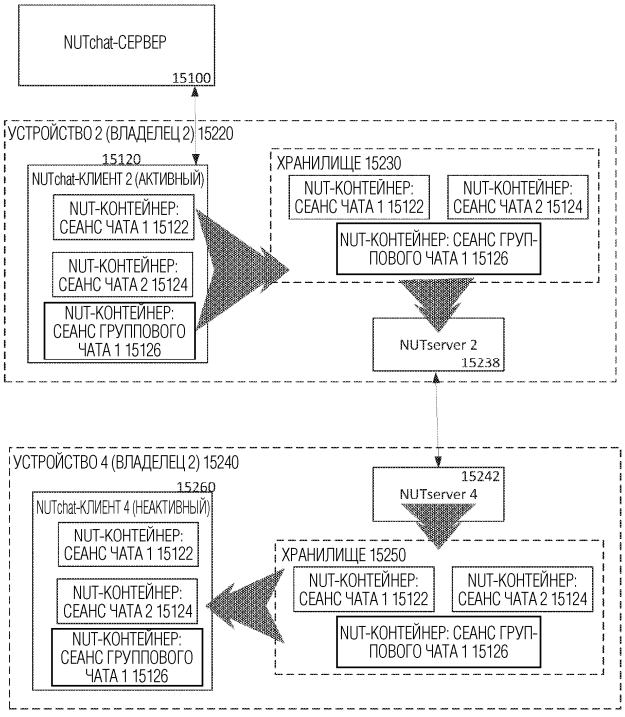
Фиг. 149



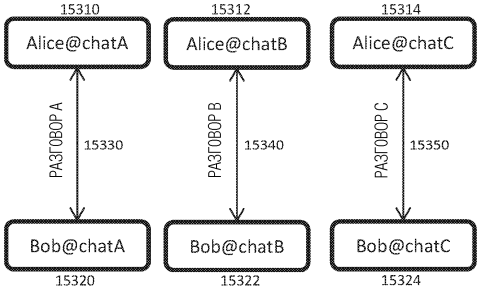
Фиг. 150



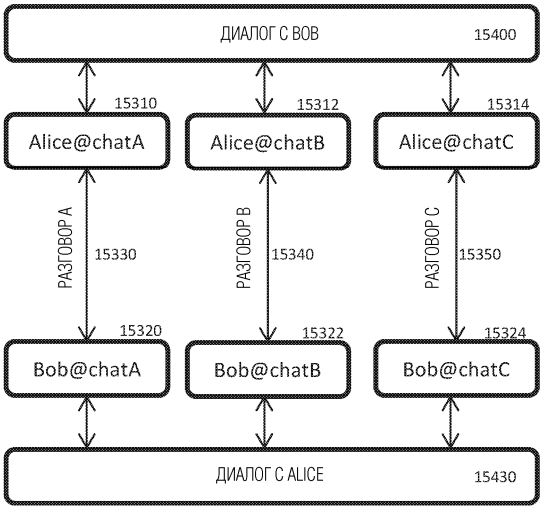
Фиг. 151



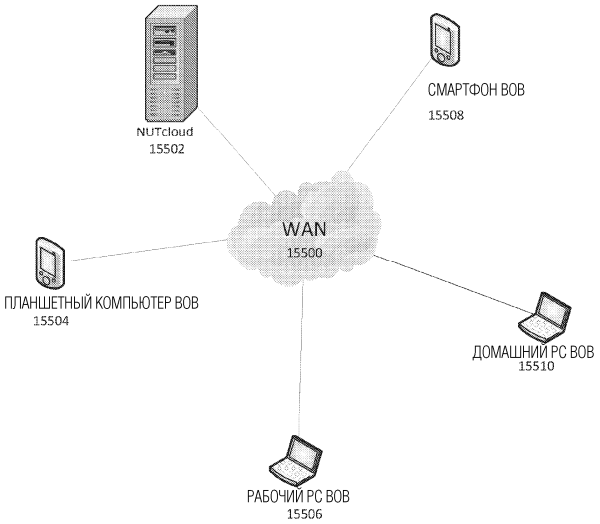
Фиг. 152



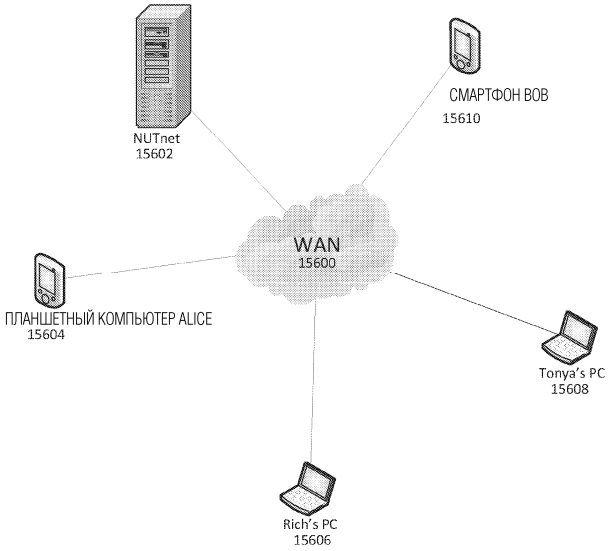
Фиг. 153



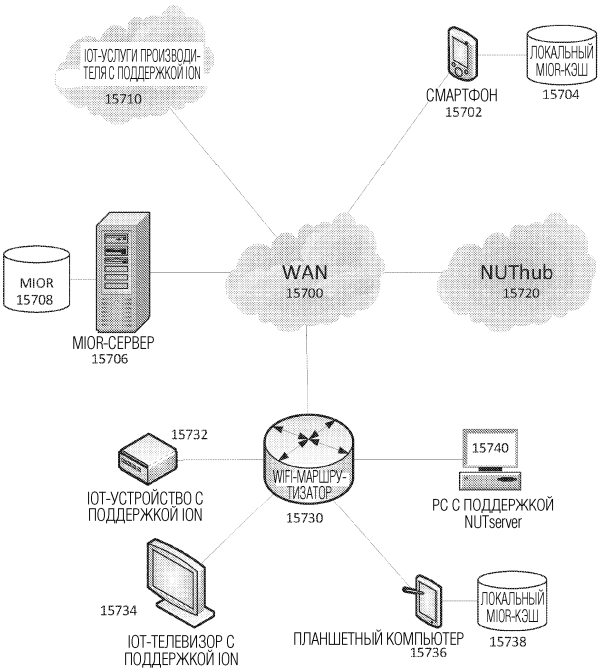
Фиг. 154



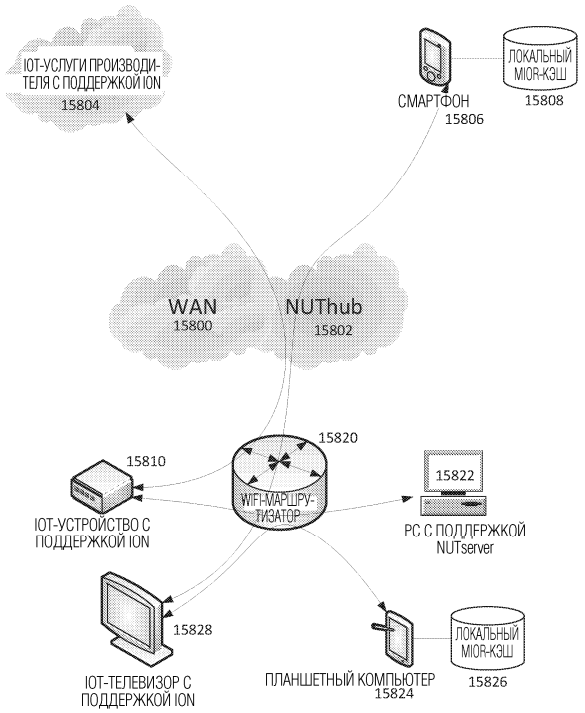
Фиг. 155



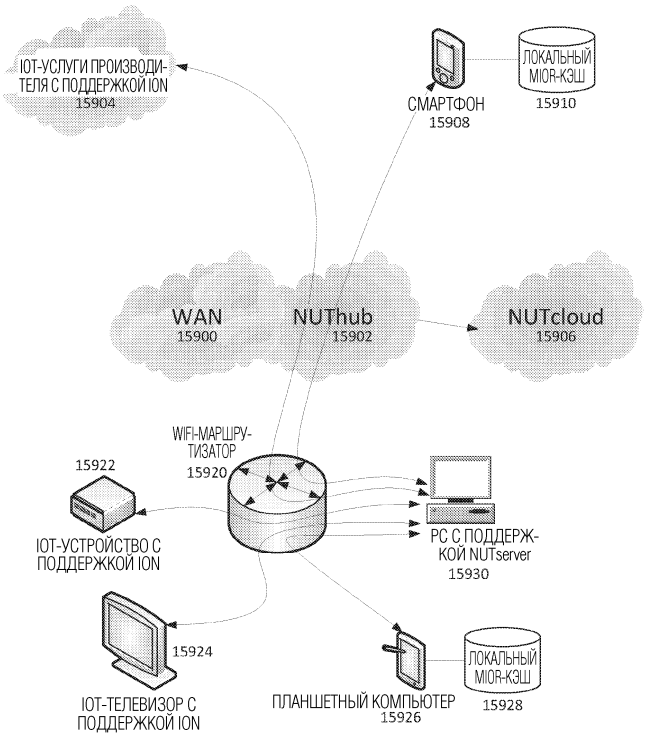
Фиг. 156



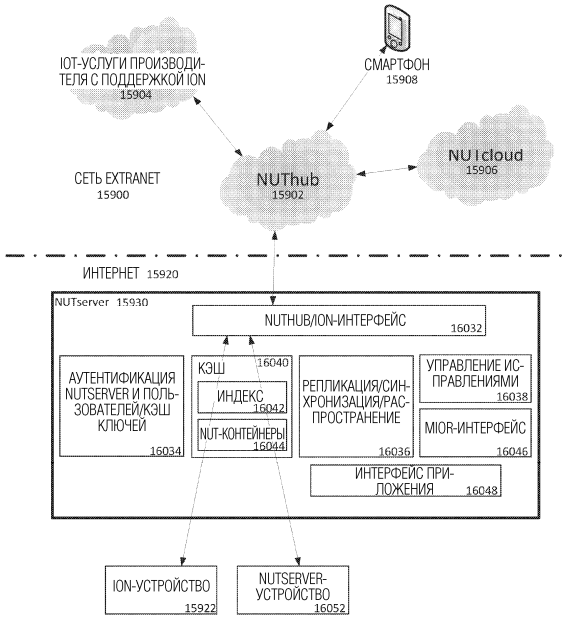
Фиг. 157



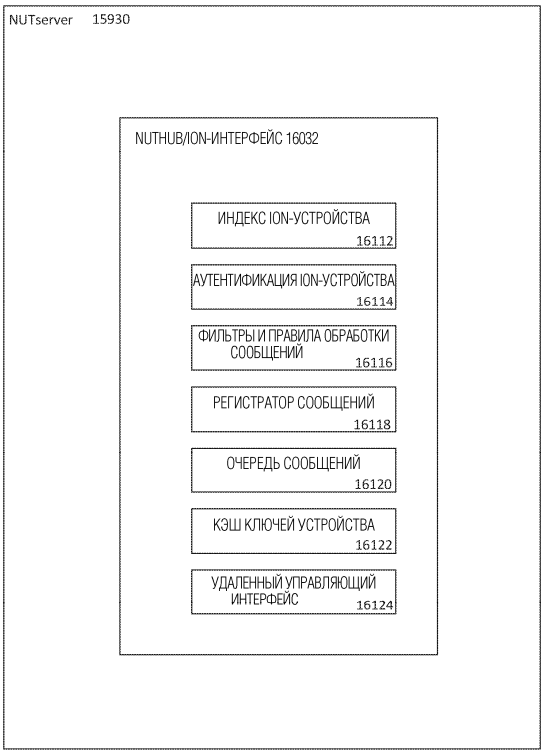
Фиг. 158



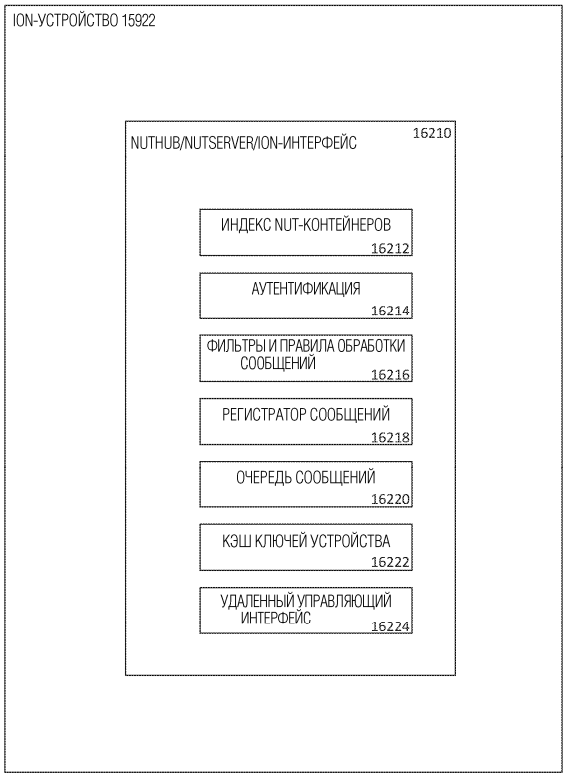
Фиг. 159



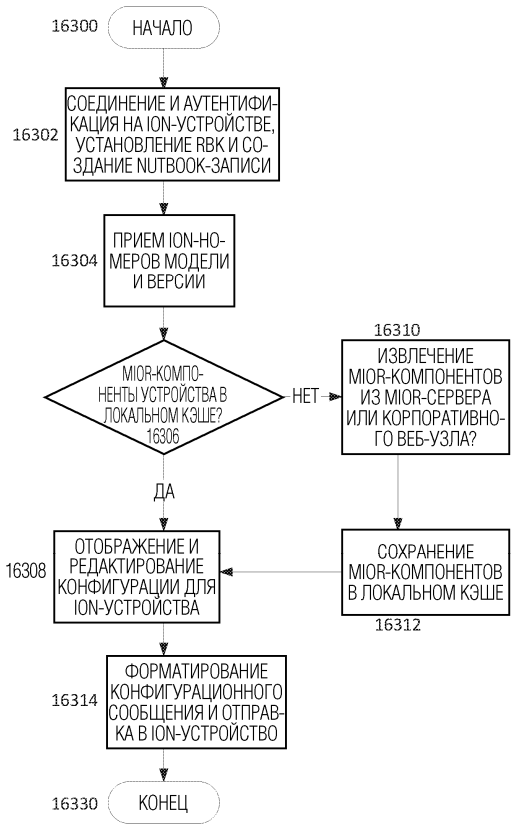
Фиг. 160



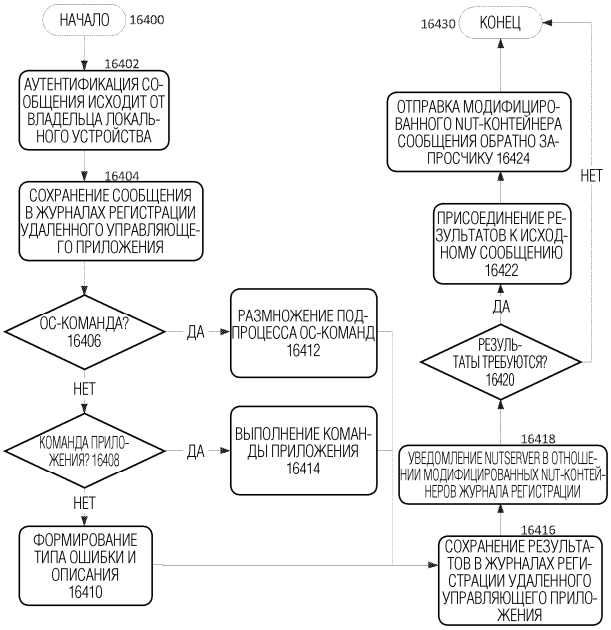
Фиг. 161



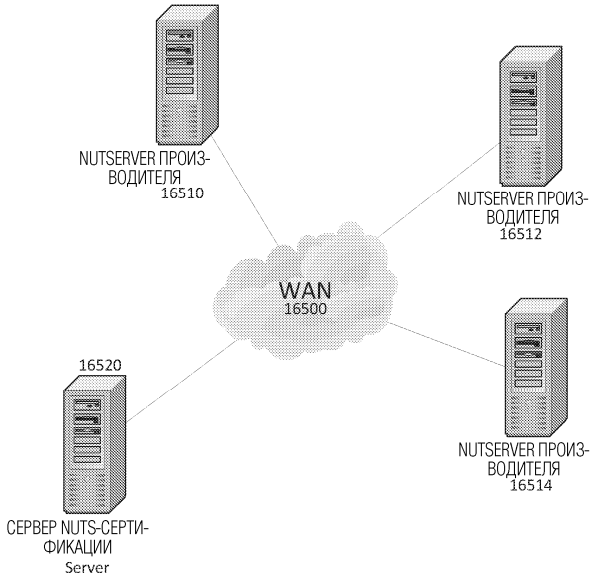
Фиг. 162



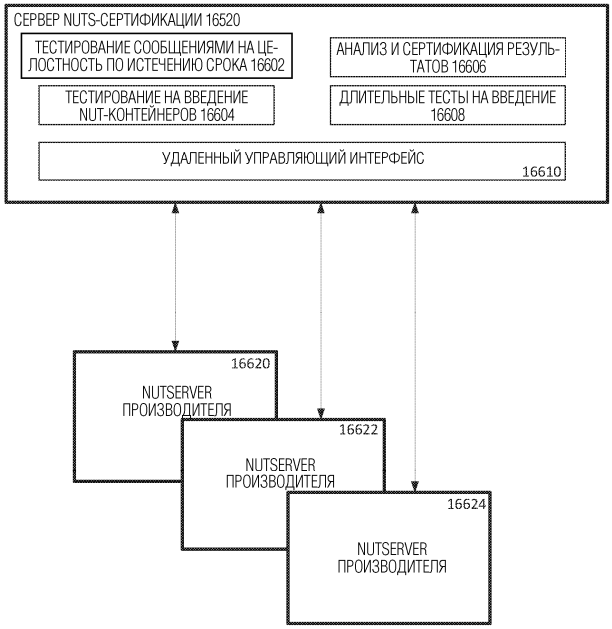
Фиг. 163



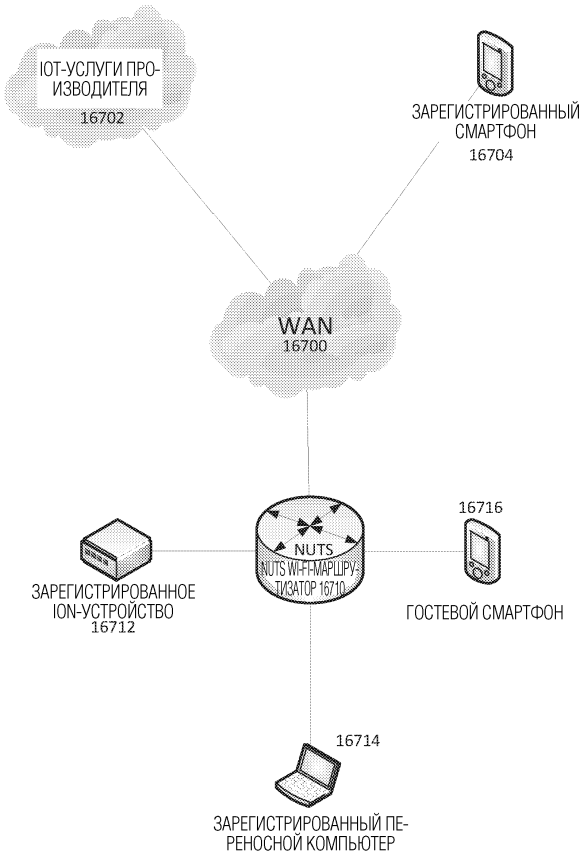
Фиг. 164



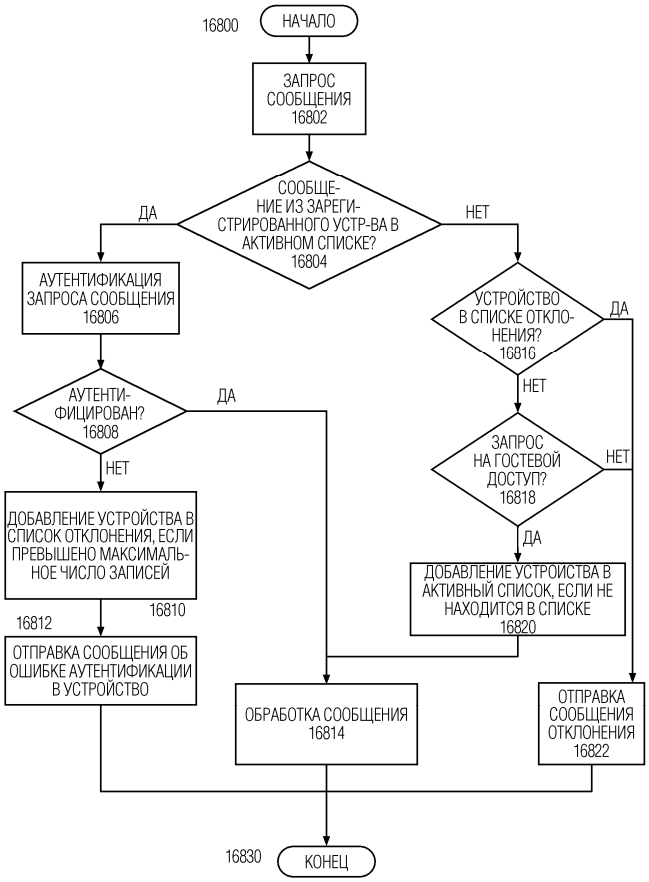
Фиг. 165



Фиг. 166



Фиг. 167



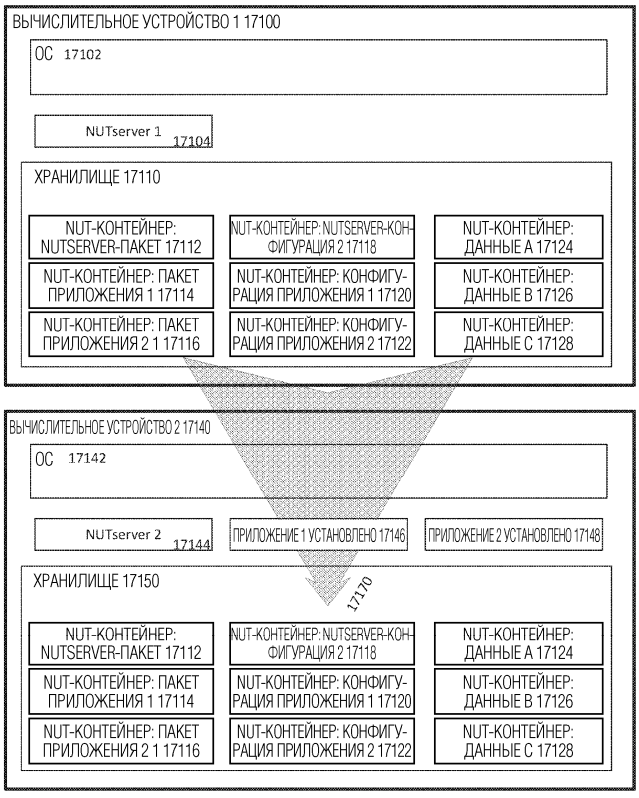
Фиг. 168

КАТЕГОРИЯ УСТРОЙСТВ	АУТЕНТИФИКАЦИЯ ПО NUT-ИДЕНТИФИКАТОРУ	ДОСТУП АДМИНИСТРАТОРА МАРШРУТИЗАТОРА
ЗАРЕГИСТРИРОВАННЫЕ	ТРЕБУЕТСЯ	ДА
IoT	ТРЕБУЕТСЯ	НЕТ
ГОСТЕВЫЕ	НЕТ	НЕТ

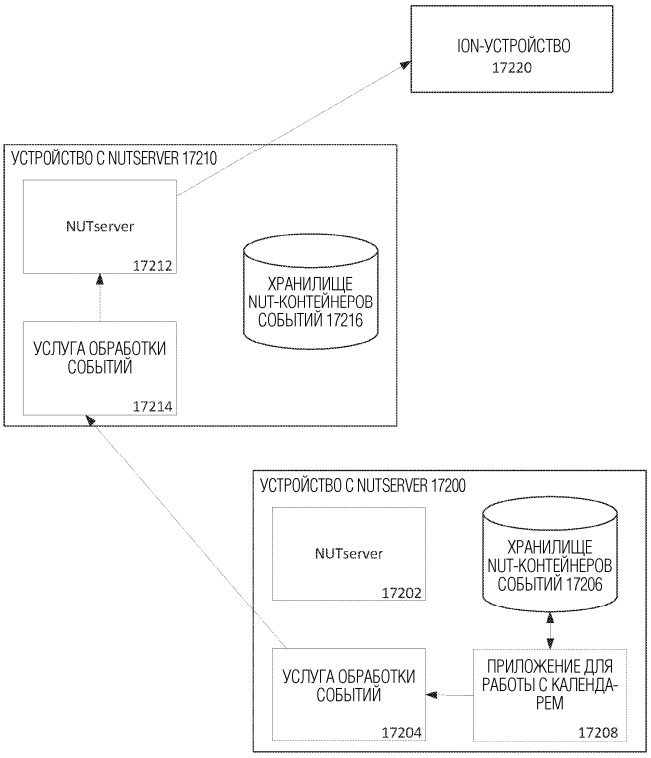
Фиг. 169

АТРИБУТЫ СОЕДИНЕНИЯ	ЗАРЕГИСТРИРОВАННОЕ УСТРОЙСТВО	IOT-УСТРОЙСТВО	ГОСТЕВОЕ УСТРОЙСТВО
ИСТЕЧЕНИЕ СРОКА ДЕЙСТВИЯ	НЕТ	НЕТ	1 ДЕНЬ
ПОЛОСА ПРОПУСКАНИЯ	ПОЛНАЯ	ПОЛНАЯ	5%
СОВОКУПНАЯ ПОЛОСА ПРОПУСКАНИЯ	НЕОГРАНИЧЕННАЯ	1 МБ/ДЕНЬ	200 МБ/ДЕНЬ
МАКСИМАЛЬНОЕ ЧИСЛО СОЕДИНЕНИЙ	100	20	10
НАЗНАЧЕНИЯ	НЕОГРАНИЧЕННЫЕ	ПРЕДВАРИТЕЛЬНЫЙ УТВЕРЖДЕННЫЙ СПИСОК	ИСПОЛЬЗОВАНИЕ ОТКРЫТОГО ФИЛЬТРА
РЕЖИМ ПЕРЕДАЧИ СООБЩЕНИЙ	НЕМЕДЛЕННЫЙ	ХРАНЕНИЕ И АНАЛИЗ	НЕМЕДЛЕННЫЙ

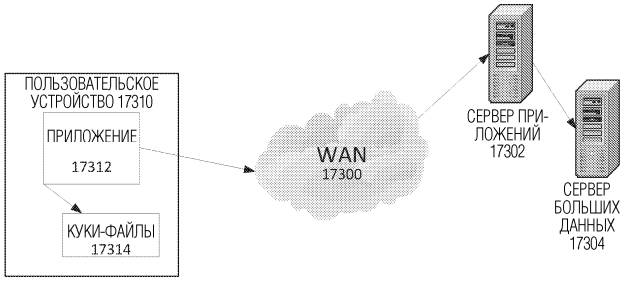
Фиг. 170



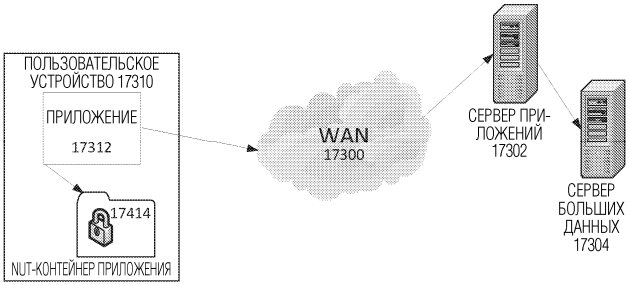
Фиг. 171



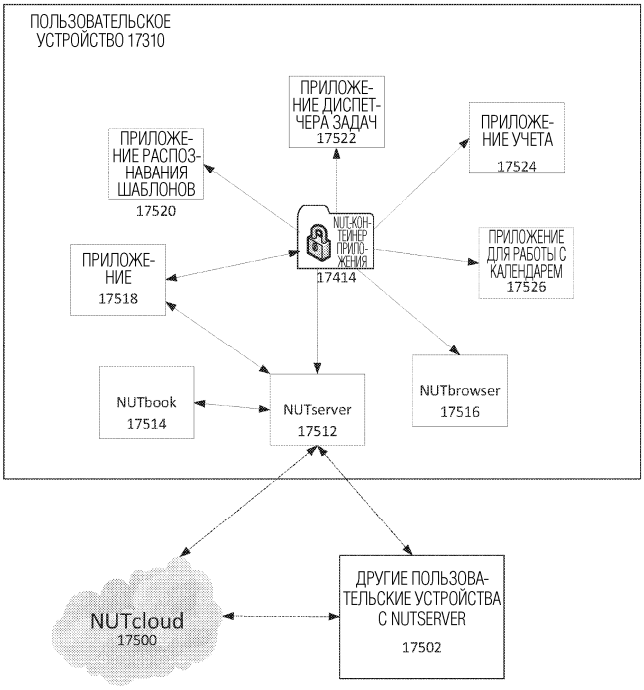
Фиг. 172



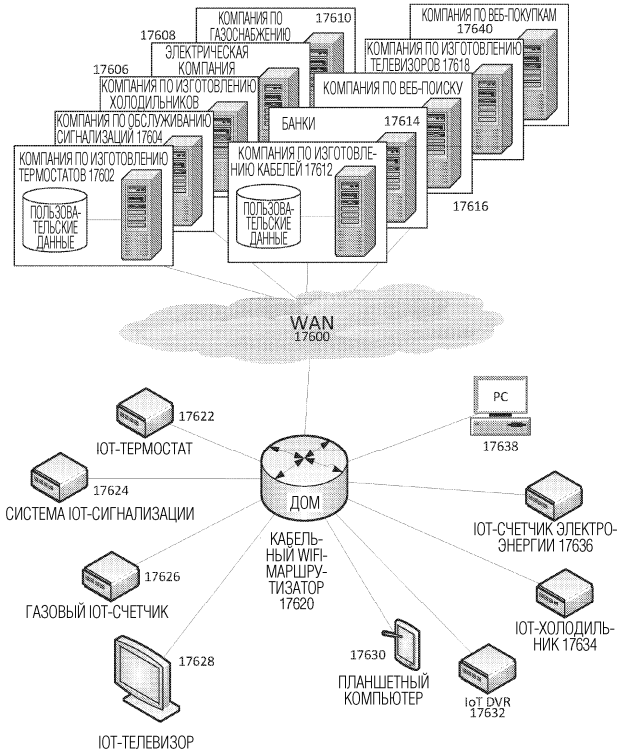
Фиг. 173



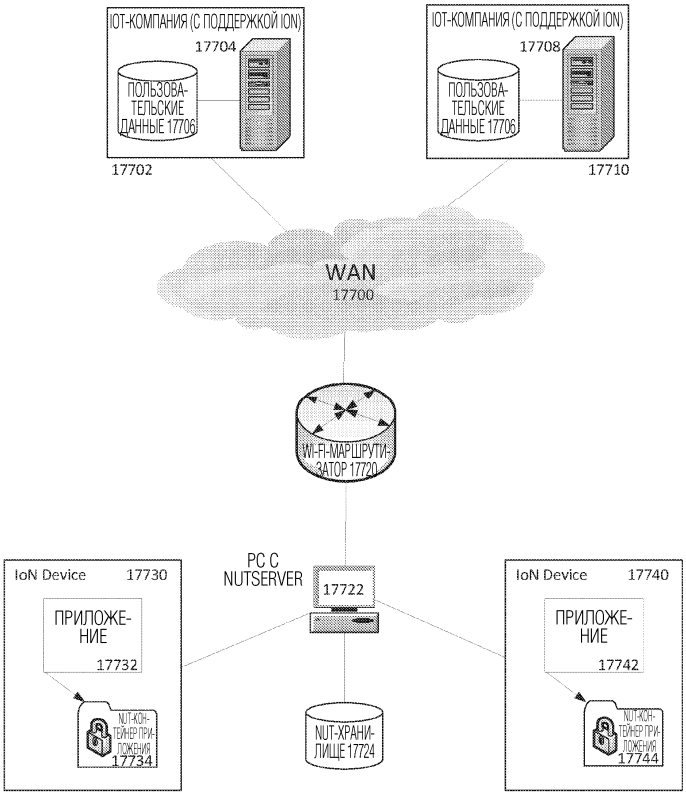
Фиг. 174



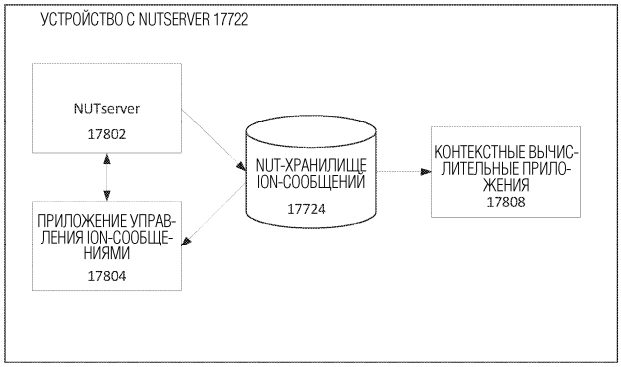
Фиг. 175



Фиг. 176



Фиг. 177



Фиг. 178