УДК 004.021 DOI: 10.18698/2541-8009-2019-04-459

# ИССЛЕДОВАНИЕ АЛГОРИТМОВ ШИФРОВАНИЯ CRYPTOAPI LINUX В КОНТЕКСТЕ РАБОТЫ С USB-ДРАЙВЕРАМИ

**Д.С. Бородин** demansen@yandex.ru
SPIN-код: 8456-7025

**П.В. Федоров** dafferrus@gmail.com
SPIN-код: 8757-0041

**МГТУ им. Н.Э. Баумана, Москва, Российская Федерация**

**Аннотация**

*Исследованы алгоритмы шифрования, предоставляемые CryptoAPI Linux с целью выбора наиболее эффективного для работы с шифрованием и расшифровкой данных на USB-накопители. Приведена классификация подобных алгоритмов, указаны их преимущества и недостатки, а также критерии оценивания алгоритмов. Выполнено исследование скоростных характеристик алгоритмов в зависимости от производительности процессора. Проведено сравнение выбранных алгоритмов с аналогичными алгоритмами по производительности. Рассмотрены основные системные вызовы CryptoAPI Linux, а также способы регистрации драйвера в операционной системе Linux и его взаимодействие с устройствами системы. Дано описание модели взаимодействия с USB-системой хранения данных.*

**Ключевые слова**

*USB-драйвер, загружаемый модуль ядра, CryptoAPI, шифрование, симметричные алгоритмы шифрования, AES, DES, Cast6, ARC4*

Поступила в редакцию 21.01.2019


Часто информация, которая находится на USB-flash-устройствах, является конфиденциальной, другие пользователи не должны иметь к ней доступ. Одним из способов ограничения доступ пользователя к информации является их шифрование. Для шифрования данных существует множество алгоритмов, однако такую процедуру необходимо применять к каждому файлу, находящемуся на устройстве. Более рациональным является подход, при котором данные автоматически шифруются при записи и расшифровываются при считывании.

Целью работы является создание драйвера, который будет шифровать данные, передаваемые на USB-flash-устройство, и расшифровывать их при считывании с устройства.

Для разработки подобного драйвера командой разработки были решены следующие задачи:

– выполнено исследование взаимодействия USB-устройств;

– реализована регистрация USB-драйвера в операционной системе (ОС) Linux;

– исследованы методы записи и чтения USB-устройств, а также методы шифрования;

– выполнено проектирование и реализация модуля ядра, отвечающего за работу драйвера.

**Структура USB-устройств.** USB-устройства в общем случае представляют собой универсальную последовательную шину (universal serial bus), которая является соединением между компьютером и несколькими периферийными устройствами. Топологически подсистема USB представляет собой четыре соединения типа «точка — точка». Связь с USB-устройством осуществляется по оконечным точкам. Оконечная точка USB может переносить данные только в одном направлении, либо со стороны хост-компьютера на устройство (такая точка называется OUT) или с устройства на хост-компьютер (такая точка называется IN). Существует четыре типа оконечных точек [1]:

**1)** ***управляющие*** — используются для обеспечения доступа к различным частям USB-устройства. Часто применяются для настройки устройства путем передачи команд и получением статусных команд;

**2)** ***прерывание*** — передают небольшие объемы данных с фиксированной частотой. Эти точки являются основным транспортным методом для USB- клавиатур и мышей;

**3)** ***поточные*** — передают большие объемы данных. Эти точки используются USB-устройствами, которые передают большие объемы данных без потерь, в том числе USB-flash-накопителями;

**4)** ***изохронная*** — способны передавать большие объемы данных, но их доставка не гарантируется. Чаще всего используются для USB-устройств, считывающие видео- или аудиопоток данных.

В системе Linux данные оконечные точки описываются структурой, в которой указывается тип точки, адрес, максимальные размер в байтах принимаемых или отправляемых пакетов.

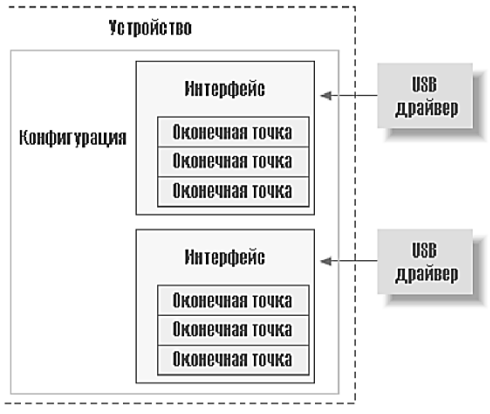


**Рис 1.** Архитектура USB-устройств

Все оконечные точки USB являются частью интерфейса. USB-интерфейсы обрабатывают только один тип соединения с устройством, таким как мышь, клавиатура, устройство хранения данных. Некоторые устройства могут иметь несколько интерфейсов, поскольку интерфейс представляет собой набор операций для выполнения ограниченных операций с устройством. Каждый драйвер USB управляет одним конкретным интерфейсом. Данное архитектурное решение демонстрирует рис. 1. В качестве примера устройства с двумя интерфейсами рассмотрим аудиоколонку. Для работы с таким устройством ОС Linux необходимо два разных драйвера для одного аппаратного устройства: для аудиопотока — один интерфейс, для кнопок на динамике — другой.

USB-интерфейсы могут иметь дополнительные параметры настройки, которые представляют собой разные наборы параметров интерфейсов. Данные параметры могут быть использованы для управления отдельными оконечными точками.

Сами USB-интерфейсы являются частью конфигурации. USB-устройства могут иметь множество конфигураций и могут переключаться между ними с целью изменения состояния устройства. Например, устройства, которые позволяют загружать в них программное обеспечение, для решения этой задачи содержат несколько конфигураций. В один момент времени может быть разрешена только одна конфигурация. ОС Linux не очень хорошо обрабатывает множественную конфигурацию USB устройства.

Таким образом, USB устройства являются довольно сложными и состоят из множества разных логических единиц. Отношения между этими частями представлены на рис. 1 и могут быть описать следующим образом:

– устройство имеет одну или более конфигураций;

– конфигурации предусматривают один или несколько интерфейсов;

– интерфейсы могут иметь оконечные точки или не иметь их вовсе;

– интерфейсы имеют один или несколько наборов параметров.

При работе с USB-flash-устройствами чаще всего определены всего две оконечные точки (поточная IN и поточная OUT). Для работы с устройством необходимо знать адреса этих точек. Интерфейс у USB-flash-устройств один (данный параметр указывается в структуре интерфейса).

**Регистрация USB-драйвера.** Прежде чем начать работу с USB-устройством, ОС Linux проводит поиск драйвера для подключенного устройства. Каждый USB-драйвер имеет специальную структуру **usb_driver** [2]. Данная структура должна быть заполнена драйвером для работы с USB-ядром. Основными полями в структуре являются [3]:

**– const char * name** — указатель на имя драйвера. Имя должно быть уникальным среди всех USB драйверов в ядре;

**– const struct usb_device_id *id_table** — указатель на таблицу, которая содержит список всех поддерживаемых драйвером устройств. Если данная структура не установлена, то функция probe никогда не вызовется;

**– int (*probe) (struct usb_interface *intf)** — указатель на зондирующую функцию в USB-драйвере. Данная функция вызывается USB-ядром системы Linux для проверки, может ли данный драйвер обработать новое устройство. Если данная функция возвращает нуль, то она подтверждает, что драйвер может обрабатывать устройство. Если же произошла ошибка инициализации драйвера или драйвер не может обработать данное устройство, то оно должно вернуть отрицательный код ошибки;

**– int (*disconnect) (struct usb_interface *intf)** — указатель на функцию отключения устройства в USB-драйвере. Данная функция вызывается USB-ядром Linux в момент удаления интерфейса из системы.

Для того чтобы зарегистрировать данную структуру в USB-ядре Linux, необходимо выполнить вызов **usb_register_driver** с указателем на данную

структуру. Это действие осуществляется в функции **probe**. Когда драйвер USB будет выгружаться, необходимо будет разрегистрировать структуру в USB-ядре. Для этого используется системный вызов **usb_deregister** с указанием структуры, которую необходимо разрегистрировать [4].

Функции **probe** и **disconnect** вызываются в контексте потока USB-узла ядра, поэтому блокировка в них допускается. В функции обратного вызова **probe** USB-драйвер должен проинициализировать все необходимые для работоспособности драйвера локальные и глобальные переменные и структуры. Следует сохранить в локальные структуры любую необходимую информацию об устройстве (адреса оконечных точек, размер буфера для обмена информацией).

**Методы записи и чтения данных с USB-flash-устройства. Интерфейс малых вычислительных систем(SCSI).** Для обмена данными с USB-flash-устройствами необходимо учитывать модель обмена информацией. Для обмена данными с USB устройством хранения данных создан интерфейс, называемый SCSI (Small Computer Systems Interface). Данный интерфейс предоставляет собой набор стандартов для связи с устройствами хранения данных. SCSI реализован в системе Linux как клиент-серверная архитектура связи, где клиентом является компьютер, а сервером устройство. Инициатор отправляет запросы команд целевому носителю, который может ответить на поступивший запрос. Для данного интерфейса конечным устройством может быть дисковый накопитель, CD-ROM, накопитель на магнитной ленте и другие устройства хранения данных. Команды определяются в блоке дескриптора команды (Command Description Block). Каждая команда определяет ту или иную операцию, которую предстоит выполнить устройству. Ответ команды может быть передан в качестве буфера данных либо в качестве блока состояния устройства [5]. Передача осуществляется с помощью блока прямого доступа. USB-устройство отправляет данные на компьютер. Данный блок принимается специальном интерфейсом SCSI, который размещает его в память прямого доступа.

Перечислим основные команды, необходимые для взаимодействия с устройством.

**Inquiry** — данная команда запрашивает у устройства структуру с информацией об устройстве. Устройство должно возвращать ответ на данную команду даже в случае неготовности носителя. Ответ приходит из буфера, полученного от носителя. Минимальная длина такого буфера составляет 36 байт. Ответ содержит информацию о носителе, его тип, версию SCSI, которую он поддерживает, количество портов у устройства и другую информацию. Однако за первыми 36 байтами в ответе может находиться специальная информация для взаимодействия с нестандартным устройством.

**Request sense** — команда запрашивает структуру, содержащую дополнительные данные. Ответ содержит расширенное состояние носителя на момент запроса к нему, содержит ли носитель данные.

**Read_Capacity** — используется для того, чтобы выяснить, какой объем данных может хранить устройство.

**Test unit ready** — данная команда запрашивает устройство, готово ли оно к передаче данных.

**Write** — является командой записи данных на устройство. В качестве данных может указываться физический адрес данных в оперативной памяти компьютера либо сами данные, однако для USB-flash-устройств хранения данных необходимо передать именно физический адрес передаваемых данных, а также необходимо указывать размер данных которые будут переданы на устройство. Для данной команды определены четыре варианта команд. Различием между командами является размер пересылаемого логического блока. После передачи командного блока на этапе пересылки данных устройство должно принять данные для записи. Устройство не должно интересоваться содержимым данных, все что необходимо для записи, номер блока и число блоков.

**Read** — команда является обратной команде write и указывает, что необходимо вернуть компьютеру данные. При передаче данных необходимо указать количество блоков для чтения. Размер блока определяется форматом команды Read_6 — блоки размером 6 байт, Read_10 — блоки размером 10 байт. Также как и в случае команды write, носитель не должен знать, что находится в запрошенных блоках.

После пересылки данных драйверу необходимо отправить контейнер статуса команды (CSW). Эта необходимость объясняется необходимостью проверки успешного получения данных от устройства. Данный блок содержит сигнатуру, тег команды, объем данных переданных на этапе передачи данных и другую служебную информацию.

Данный интерфейс указывает модель взаимодействия с устройством. USB-flash-устройства поддерживают данный стандарт передачи и называются устройствами SCSI устройствами [6].

Рис. 2 отображает основные способы взаимодействия драйвера с USB-устройством. В самом начале необходимо передать контейнер набора команд. (**B**ulk **C**ommand **B**lock). После передачи данного блока необходимо сообщить адрес блока памяти, которое будет использоваться для передачи данных между устройством и компьютером. После такой подготовки можно отправлять команды USB-устройству. Необходимо придерживаться данной модели взаимодействия, так как в случае отклонения от модели устройство не будет обрабатывать данные.

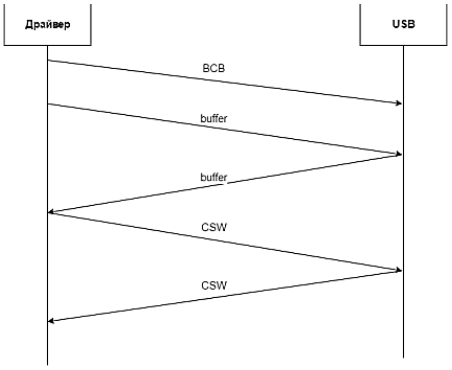


**Рис. 2.** Отображение взаимодействия с USB-устройством хранения данных

Кроме того, возможна работа через приложение с SCSI-командами (правый блок на рис. 3), которое будет формировать необходимые запросы из пользовательской программы. Данный способ также является распространенным, но применяется лишь для специальных устройств.

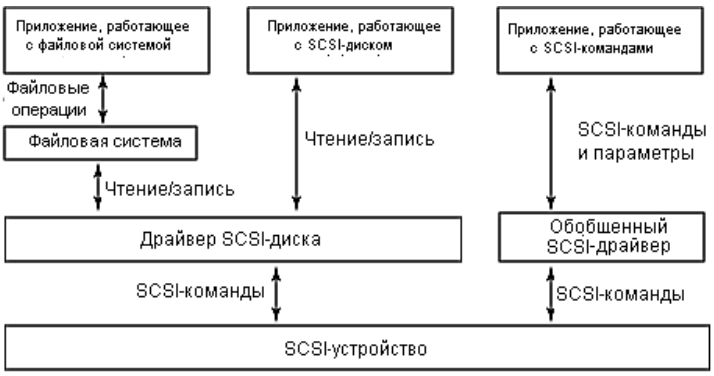


**Рис. 3.** Способы взаимодействия со SCSI-устройством

В данной работе используется метод взаимодействия через файловую систему ОС Linux, поскольку этот вариант предусматривает работу с данными на уровне системы, а не на уровне драйвера.

Обмен информацией с USB-устройством может осуществляться двумя способами.

**Блок запросов USB (URB).** Первый способ заключается в использовании **блоков запроса USB.** В ядре Linux взаимодействует со всеми устройствами USB с помощью **URB (USB request block,** блок запросов USB). Данный блок запроса описывается структурой struct urb. URB используется для передачи или приема данных в заданной оконечной точке USB или из нее на заданное USB-устройство в асинхронном режиме. В зависимости от потребностей драйвер USB-устройства выделяет для одной оконечной точки любое количество блоков запросов USB или повторно использует один блок запросов USB для множества разных оконечных точек. Каждая оконечная точка в устройстве может обрабатывать очередь блоков запросов. Так, перед тем как очередь опустеет, к одной оконечной точке может быть отправлено множество этих запросов USB. Жизненный цикл блоков запросов USB выглядит следующим образом:

– создание блока запроса USB-драйвером;

– назначение в определенную оконечную точку заданного USB-устройства;

– передача драйвером USB-устройства в USB-ядро;

– передача USB-ядром в заданный драйвер контроллера USB-узла для заданного устройства;

– обработка драйвером контроллера USB-узла, который выполняет передачу по USB в устройство;

– после завершения работы с блоком запросом USB-драйвер контроллера USB-узла уведомляет драйвер USB-устройства.

Блоки запросов USB также могут быть отменены в любое время драйвером, который передал их, или ядром USB, если устройство удалено из системы. Эти блоки запросов создаются динамически. Они позволяет передавать данные с большим объемом, чем максимальный размер данных, принимаемый по указанной оконечной точке. Это связано с тем, что USB-ядро ОС Linux позволяет

разбить буфер на маленькие блоки данных. Гораздо быстрее поместить большой блок данных в один блок запросов USB и предоставить его контроллеру USB-узла, разделить его на буферы малого размера USB-ядром, чем отправить эти буферы последовательно. Таким образом осуществляется прием данных с USB устройства. Рассмотрим структуру блока запросов USB для понимания взаимодействия с ним:

**– struct usb_device *dev** — указатель на структуру struct usb_device, в которую будет послан этот urb. Эта переменная должна быть проинициализирована USB-драйвером перед тем, как блок запроса может быть отправлен в ядро USB;

**– unsigned int pipe** — адрес оконечной точки, в которую будет передан данный urb. Чтобы установить поля этой структуры, драйвер использует специальные функции получения адреса оконечной точки текущей конфигурации с текущим интерфейсом;

**– unsigned int transfer_flags** — эта переменная может быть установлена в несколько различных битовых значений в зависимости от предназначения буфера, который использует USB-драйвер;

**– void *transfer_buffer** — указатель на буфер, который будет использоваться при передаче данных в устройство (для ВЫХОДНОГО блока) или при получении данных из устройства (для ВХОДНОГО блока). Для того чтобы хост-контроллер правильно получал доступ к этому буферу, он должен быть создан вызовом **kmalloc**, а не на стеке или статически;

**– dma_addr_t transfer_dma** — буфер для использования для передачи данных в USB-устройство с помощью DMA;

**– int transfer_buffer_length** — длина буфера, на который указывает переменная transfer_buffer или transfer_dma (только одна из них может быть использована для блока запросов USB);

**– unsigned char *setup_packet** — указатель на установочный пакет для управляющего блока запросов USB. Он передается перед данными в буфере передачи. Эта переменная используется только в управляющих urb;

**– dma_addr_t setup_dma** — DMA-буфер для установочного пакета управляющего urb. Он передается перед данными в обычном буфере передачи. Эта переменная используется только для управляющих urb;

**– usb_complete_t complete** — указатель на завершающую функцию обработки, которая вызывается USB-ядром, когда urb полностью передан или при возникновении ошибки с urb. В этой функции USB-драйвер может проверить urb, освободить его или использовать повторно для другой передачи;

**– void *context** — указатель на данные blob (Binary Large Object, большой двоичный объект), который может быть установлены USB-драйвером. Он может быть использован в завершающем обработчике, когда urb возвращается в драйвер;

**– int actual_length** — после завершения urb эта переменная равна фактической длине данных или отправленных urb или полученных urb;

**– int status** — после завершения urb или его обработки USB-ядром эта переменная содержит текущий статус urb. Для безопасного получения доступа к данным urb и его состояния используется функция обработчика завершения

urb. Это ограничение является защитой от состояний гонок (борьба двух процессов за разделяемый ресурс), которые происходят в то время, когда блоки запросов USB обрабатывается USB-ядром.

***Преимущества*:** большая скорость передачи, возможность передавать большие объемы данных за одну передачу.

***Недостатки*:** сложность инициализации структуры.

**Системные вызовы.** Передачу данных на USB-устройство можно выполнить с использованием двух функций **usb_bulk_msg** и **usb_control_msg.** Эти функции дают возможность передать данные без заполнения сложных структур. Данные передаются в указанное устройство с помощью urb, которое создается USB-ядром, затем USB-ядро ожидает завершение передачи, прежде чем передать управление обратно вызывающему процессу. Если функция завершилась успешно, то возвращается нуль, если же произошла ошибка, то возвращается отрицательный код ошибки. При успешном выполнении один из параметров функции, который отображает количество переданных устройству байт, устанавливается в соответствующее значение. Однако такой подход отправки данных в устройство или из него не может быть отменен, поэтому стоит учитывать этот факт при вызове функции **disconnect**.

***Преимущества*:** легкость инициализации отправки данных (не нужно заполнять структуры).

***Недостатки:*** низкая скорость передачи, невозможность указать формат отправляемых данных.

В результате выполненного анализа был выбран способ передачи данных с помощью (блоков запросов USB) **urb**, поскольку данная структура является гибкой в плане настроек передачи данных и обеспечивает быструю передачу этих данных.

**Методы шифрования и расшифровки.** Для шифрования данных необходимо выбрать алгоритм. CryptoAPI — набор методов и функций для шифрования и расшифровки данных, который предоставляет нам обширный выбор алгоритма. Алгоритмы шифрования подразделяют на два типа — ***симметричные* и *ассиметричные.***

Симметричные алгоритмы используют один и тот же ключ для шифрования и расшифровки. Ассиметричные имеют два ключа: открытый (для шифровки, известный всем) и закрытый ключ (для расшифровки известный только расшифровщику). Кроме того, алгоритмы подразделяют на блочные и поточные. Блочные алгоритмы шифруют данные по блокам данных, что является ограничением для шифрования данных. Поточные алгоритмы шифруют данные любой длины, основываясь на уже зашифрованных данных. Защищенность данных, которые зашифрованы алгоритмом шифрования, зависит от криптостойкостьи самого алгоритма (способность алгоритма противостоять попыткам взлома).

Рассмотрим распространенные алгоритмы шифрования.

1. **DES (Data Encrypt Standard)** — симметричный алгоритм для блочного шифрования. Шифруемая информация получается путем побитового сложения с некоторой постоянной функцией. Для работы выполняется начальная пере-

становка шифруемого блока, затем блок делиться на два подблока (левый и правый) одинакового размера. Левый блок остается неизменным, байты правого подблока побитового складываются по модулю на основе ключей каждого своего байта. Затем левый и правый блок меняются местами. Данная операция повторяется 16 раз. После блоки объединяются в один и осуществляется конечная перестановка. Таким образом получается шифрованный блок данных [6].

***Преимущества*:** размер блока 8 байт — самый маленький блок среди блочных алгоритмов, быстрота шифрования, малое количество операций.

***Недостатки*:** маленький размер ключа, алгоритм устаревший.

2. **DES3** — алгоритм блочного шифрования, является модификацией алгоритма DES, отличается от предыдущего размером ключа, который равен 21 байту и числу операций. Отличием алгоритма от DES заключается в том, что ключ делится на три части и с каждой частью ключа производиться шифрование алгоритмом DES [7].

***Преимущества*:** размер блока 8 байт, криптостойкость, размер ключа.

***Недостатки*:** невысокая скорость шифрования (в 3 раза медленнее DES).

3. **AES (Advanced Encryption Standard)** — алгоритм блочного шифрования, является новым стандартом блочным алгоритмов шифрования [8]. Существует три вида AES различающихся размером шифро-ключа. Перед началом работы полученный ключ преобразовывается для работы с данными и увеличивается в размере в 10 раз. Для шифрования информации алгоритму необходим блок данных размером 16 байт. Затем выполняются простые операции по преобразованию данного блока. Первой операцией является подмена одного байта на другой байт, получаемый из константной таблицы AES. Затем к полученному блоку применяется сдвиг байт в каждой строке. После чего к блоку данных применяется перемешивание столбцов, основанное на матричном умножении матрицы константы и блока данных. Эти действия повторяются 10 раз, на последней итерации перестановка столбцов не выполняется. Для расшифровки данных выполняется последовательность, описанная ранее, но в обратную сторону.

***Преимущества*:** быстрота шифрования и расшифровки, размер ключа, криптостойкость.

***Недостатки*:** размер блока 16 байт.

4. **Cast6 (Cast 256)** — блочный алгоритм шифрования. Блок данных в 16 байт делится на четыре части, по 4 байта каждый. Затем каждый бит предпоследнего блока складывается по модулю с битами полученными из преобразования последнего блока, сам последний блок не изменяется. После данной операции блоки перемешиваются в определенном порядке. Такая операция производится 48 раз. Ключ в данном алгоритме определяет поведение функции преобразования последнего блока данных [9].

***Преимущества*:** возможность расширения ключа, размер ключа может варьироваться.

***Недостатки*:** медленная скорость шифрования, большие требования к оперативной и энергозависимой памяти.

5. **ARC4** — потоковый алгоритм, широко применяющийся в различных системах защиты информации в компьютерных сетях. Шифр разработан компанией RSA security. Данный алгоритм работает на основе генератора псевдослучайных битов. Ключ в данной ситуации задает настройки генератор, который необходим для побитового сложения байта данных с генерируемыми байтами. Для расшифровки производиться регенерация генератора [10].

***Преимущества*:** высокая скорость работы, переменный размер ключа.

***Недостатки*:** уязвим к взлому, используются неслучайные ключи.

При реализации драйвера был выбран алгоритм шифрования AES, так как этот алгоритм является одним из защищенных и быстрых алгоритмов. CryptoAPI системы Linux позволяет реализовать шифрование данных с оптимальной скоростью. Также в силу того, что алгоритм является блочным, он предоставляет возможность записи и чтения одновременно нескольких файлов.

**Литература**


[1] Axelson J. USB mass storage: designing and programming devices and embedded hosts. Lakeview Research, 2006.

[2] Corbet J., Rubini A. Linux device drivers. O'Reilly Media, 2005.

[3] Linux source code: drivers. *elixir.bootlin.com: веб-сайт*. URL: http://lxr.free-electrons.com/source/drivers (дата обращения: 12.12.2018).

[4] Venkateswaran S. Essential Linux device drivers. Prentice Hall, 2008.

[5] Джонс М. Анатомия подсистемы SCSI в Linux. *ibm.com: веб-сайт*. URL: http://www.ibm.com/developerworks/ru/library/l-scsi-subsystem/index.html (дата обращения: 12.12.2018).

[6] Satran J., Meth K., Sapuntzakis C., et al. Internet small computer systems interface (iSCSI). RFC 3720. The Internet Society, 2003.

[7] Grabbe J.O. The DES algorithm illustrated. *Laissez Faire City Times*, 1992, vol. 2, no. 28, pp. 12–15.

[8] Daemen J., Rijmen V. The design of Rijndael. Springer, 2002.

[9] CAST-6 encryption standard. *tools.ietf.org: веб-сайт*. URL: https://tools.ietf.org/html/rfc2612 (дата обращения: 12.12.2018).

[10] Ball B. Cryptography with ARC4 LPC2148. *opencore.eesc.usp.br: веб-сайт*. URL: http://www.opencore.eesc.usp.br/ricardo/Antena/nrf24l01_tutorial_4.pdf (дата обращения: 12.12.2018).



**Бородин Дмитрий Сергеевич** — студент кафедры «Программное обеспечение ЭВМ и информационные технологии» МГТУ имени Н.Э. Баумана, Москва, Российская Федерация.

**Федоров Павел Вячеславович** — студент кафедры «Программное обеспечение ЭВМ и информационные технологии» МГТУ имени Н.Э. Баумана, Москва, Российская Федерация.

**Научный руководитель** — Рудаков Игорь Владимирович, кандидат технических наук, доцент, заведующий кафедрой «Программное обеспечение ЭВМ и информационные технологии» МГТУ имени Н.Э. Баумана, Москва, Российская Федерация.

# THE STUDY OF THE CRYPTOAPI LINUX ENCRYPTION ALGHORITMS IN THE CONTEXT OF WORK WITH THE USB-DRIVERS

**P.V. Borodin** demansen@yandex.ru
SPIN-code: 8456-7025

**D.S. Fedorov** dafferrus@gmail.com
SPIN-code: 8757-0041

**Bauman Moscow State Technical University, Moscow, Russian Federation**

**Abstract**

*The encryption algorithms provided by CryptoAPI Linux for choosing the most efficient to work with encryption and decryption of data on USB-drives are investigated. A classification of such algorithms is given, their advantages and disadvantages, as well as criteria for algorithms estimation, are indicated. The study of the algorithms speed characteristics depending on the performance of the processor has been carried out. A comparison of the selected algorithms with similar algorithms by performance has been conducted. The main system calls of CryptoAPI Linux are considered, as well as ways to register the driver in the Linux operating system and its interaction with system devices. A description of the model of interaction with USB storage system is given.*

**Keywords**

*USB driver, loadable kernel module, CryptoAPI, encryption, symmetric encryption algorithms, AES, DES, Cast6, ARC4*

Received 21.01.2019



## References

[1] Axelson J. USB mass storage: designing and programming devices and embedded hosts. Lakeview Research, 2006.
[2] Corbet J., Rubini A. Linux device drivers. O'Reilly Media, 2005.
[3] Linux source code: drivers. *elixir.bootlin.com: website*. URL: http://lxr.free-electrons.com/source/drivers (accessed: 12.12.2018).
[4] Venkateswaran S. Essential Linux device drivers. Prentice Hall, 2008.
[5] Johns M. Anatomiya podsistemy SCSI v Linux [Anatomy of SCSI system in Linux]. *ibm.com: website* (in Russ.). URL: http://www.ibm.com/developerworks/ru/library/l-scsi-subsystem/index.html (accessed: 12.12.2018).
[6] Satran J., Meth K., Sapuntzakis C., et al. Internet small computer systems interface (iSCSI). RFC 3720. The Internet Society, 2003.
[7] Grabbe J.O. The DES algorithm illustrated. *Laissez Faire City Times*, 1992, vol. 2, no. 28, pp. 12–15.
[8] Daemen J., Rijmen V. The design of Rijndael. Springer, 2002.
[9] CAST-6 encryption standard. *tools.ietf.org: website*. URL: https://tools.ietf.org/html/rfc2612 (accessed: 12.12.2018).
[10] Ball B. Cryptography with ARC4 LPC2148. *opencore.eesc.usp.br: website*. URL: http://www.opencore.eesc.usp.br/ricardo/Antena/nrf24l01_tutorial_4.pdf (accessed: 12.12.2018).

**Borodin D.S.** — Student, Department of Computer Software and Information Technologies, Bauman Moscow State Technical University, Moscow, Russian Federation.

**Fedorov P.V.** — Student, Department of Computer Software and Information Technologies, Bauman Moscow State Technical University, Moscow, Russian Federation.

**Scientific advisor** — Rudakov I.V., Cand. Sc. (Tech.), Assoc. Professor, Head of the Department of Computer Software and Information Technologies, Bauman Moscow State Technical University, Moscow, Russian Federation.