СОДЕРЖАНИЕ

Введение ……... 4
1. Краткий обзор и анализ современных алгоритмов шифрования 11
1.1 Классификация криптоалгоритмов ……...…. 11
1.2 Сеть Фейштеля как популярный атрибут блочных шифров … 19
1.3 Стандарт шифрования данных DES и AES ...……...….. 24
1.4 Схемы построения широко распространенных алгоритмов шифрования ……. 26
1.4.1 Обзор симметричных криптосистем ……...…… 27
1.4.1.1 Шифр MARS ….…...……..…….. 28
1.4.1.2 Шифр Serpent ……..…… 30
1.4.1.3 Шифр TwoFish .....……..……. 31
1.4.1.4 Шифр Rijndael ……..…… 33
1.4.1.5 Стандарт шифрования данных ГОСТ 28147-89 ……... 34
1.4.2 Обзор ассиметричных криптосистем ……..…… 38
1.4.2.1 Система с открытым ключом RSA …… 38
1.4.2.2 Криптосистема Эль-Гамаля …… 40
1.4.2.3 Криптосистемы на основе эллиптических уравнений 41
1.5 Сравнительные характеристики алгоритмов шифрования … 42
1.6 Наиболее устойчивые к взлому алгоритмы шифрования … 43
1.7 Идентификация и аутентификация пользователей …… 44
1.8 Электронно-цифровая подпись …… 48
1.8.1 Электронно-цифровая подпись при помощи симметричных криптосистем …… 49
1.8.2 Электронно-цифровая подпись при помощи асимметричных криптосистем …… 51
1.9 Хеширование паролей ……... 52
1.10 Управление криптографическими ключами ……. 57
1.10.1 Симметричная методология …… 58
1.10.2 Асимметричная методология …… 59
2. Проектирование модифицированного алгоритма RC6 …… 67

2.1 Стандартный вариант шифрования RC6 ……. 68
2.1.1 Процедура шифрования стандартного RC6 …… 70
2.1.2 Процедура расшифрования стандартного RC6 …… 72
2.1.3 Процедура генерации ключа стандартного RC6 …… 73
2.2 Модифицированный вариант шифрования RC6 …… 76
2.2.1 Процедура шифрования модифицированного RC6 …… 78
2.2.2 Процедура расшифрования модифицированного RC6 … 80
2.2.3 Процедура генерации ключа модифицированного RC6 82
3. Практическая реализация и исследование криптостойкости модифицированного RC6 …… 87
3.1 Блок-схемы процессов шифрующей программы FileCoder на основе RC6 ……... 87
3.1.1 Блок-схемы процесса шифрования файла …… 89
3.1.2 Блок-схемы процесса расшифрования файла …….. 96
3.1.3 Блок-схемы процесса подбора пароля ……. 104
3.2 Тестирование и исследование технологии шифрующей программы FileCoder на основе RC6 ……. 112
3.2.1 Шифрование/расшифрование файлов …….. 113
3.2.2 Подбор пароля в файлах …… 114
Заключение …….. 117
Библиография ……... 120
Приложение. Описание работы шифрующей программы FileCoder . 122

ВВЕДЕНИЕ

Научно-техническая революция в последнее время приняла грандиозные масштабы в области информатизации общества на базе современных средств вычислительной техники, связи, а также современных методов автоматизированной обработки информации. Применение этих средств и методов приняло всеобщий характер, а создаваемые при этом информационно-вычислительные системы и сети становятся глобальными как в смысле территориальной распределенности, так и в смысле широты охвата в рамках единых технологий процессов сбора, передачи, накопления, хранения, поиска, переработки информации и выдачи ее для использования.

Объем циркулирующей в обществе информации стабильно возрастает. Популярность всемирной сети Интернет в последние годы способствует удвоению информации каждый год. Произошедшие за последнее десятилетие изменения можно охарактеризовать следующим образом:

- объемы обрабатываемой информации возросли на несколько порядков;
- информация используется как обозначение специального товара, который можно прибрести, продать, обменять на что-то другое. При этом стоимость информации часто превосходит в сотни и тысячи раз стоимость компьютерной системы, в которой она находится;
- характер обрабатываемых данных стал чрезвычайно многообразным и более не сводится к исключительно текстовым данным;
- субъектами информационных процессов теперь являются не только люди, но и созданные ими автоматические системы, действующие по заложенной в них программе.

В современном мире информационный ресурс стал одним из наиболее мощных рычагов экономического развития. Владение информацией необходимого качества в нужное время и в нужном месте является залогом успеха в любом виде хозяйственной деятельности. Монопольное обладание определенной информацией оказывается зачастую решающим преимуществом

в конкурентной борьбе и предопределяет, тем самым, высокую цену «информационного фактора», который просто катастрофически нуждается в своей защите.

В 60-х и частично в 70-х годах проблема защиты информации решалась достаточно эффективно применением в основном организационных мер. К ним относились прежде всего режимные мероприятия, охрана, сигнализация и простейшие программные средства защиты информации. Эффективность использования указанных средств достигалась за счет концентрации информации на вычислительных центрах, как правило, автономных, что способствовало обеспечению защиты относительно малыми средствами.

Проблема обеспечения необходимого уровня защиты информации оказалась весьма сложной, требующей для своего решения не просто осуществления некоторой совокупности научных, научно-технических и организационных мероприятий и применения специфических средств и методов, а создания целостной системы организационных мероприятий и применения специфических средств и методов по защите информации.

Актуальность и важность проблемы обеспечения безопасности информационных технологий обусловлены следующими причинами:

- резкое увеличение вычислительной мощности современных компьютеров при одновременном упрощении их эксплуатации;
- резкое увеличение объемов информации, накапливаемой, хранимой и обрабатываемой с помощью компьютеров и других средств автоматизации;
- вычислительные «способности» современных компьютеров подняли на совершенно новый уровень как возможности по реализации шифров, ранее немыслимых из-за своей высокой сложности, так и возможности аналитиков по их взлому.
- сосредоточение в единых базах данных информации различного назначения и различной принадлежности;
- высокие темпы роста парка компьютеров, находящихся в эксплуатации в самых разных сферах деятельности;

- резкое расширение круга пользователей, имеющих непосредственный доступ к вычислительным ресурсам и массивам данных;
- бурное развитие программных средств, не удовлетворяющих даже минимальным требованиям безопасности;
- повсеместное распространение сетевых технологий и объединение локальных сетей в глобальные;
- развитие глобальной сети Интернет, практически не препятствующей нарушениям безопасности систем обработки информации во всем мире.

Поэтому вполне естественно возникает необходимость в защите информации от несанкционированного проникновения лиц, не имеющих к ней доступа. Информацией можно считать и отдельный файл, и базу данных, и одну запись в ней, и целиком программный комплекс. И все эти объекты могут подвергнуться и подвергаются атакам со стороны нарушителей. Поскольку информация может быть очень ценной или особо важной, возможны разнообразные злонамеренные действия по отношению к компьютерным системам, хранящим, обрабатывающим или передающим такую информацию, которые можно охарактеризовать как «атака на информацию».

Прежде всего важны экономические потери от последствия «атак на информацию», которые заключаются в следующем:

- раскрытие коммерческой информации может привести к серьезным прямым убыткам на рынке.
- известие о краже большого объема информации обычно серьезно влияет на репутацию фирмы, приводя косвенно к потерям в объемах торговых операций.
- фирмы-конкуренты могут воспользоваться кражей информации, если та осталась незамеченной, для того чтобы полностью разорить фирму, навязывая ей фиктивные либо заведомо убыточные сделки.

Таким образом, основными задачами защиты информации является:

- обеспечение конфиденциальности информации;
- обеспечение целостности информации;
- обеспечение достоверности информации;

- обеспечение оперативности доступа у информации;

Исследовательским центром DataPro Research в 1998 году были получены статистические данные об атаках на информацию [5. С.10-12]. Основные причины повреждений электронной информации распределились следующим образом: неумышленная ошибка человека – 52% случаев, умышленные действия человека – 10% случаев, отказ техники – 10% случаев, повреждения в результате пожара – 15% случаев, повреждения водой – 10% случаев. Как видим, каждый десятый случай повреждения электронных данных связан с компьютерными атаками.

Кто был исполнителем этих действий: в 81% случаев – текущий кадровый состав учреждений, только в 13% случаев – совершенно посторонние люди, и в 6% случаев – бывшие работники этих же учреждений. Доля атак, производимых сотрудниками фирм и предприятий, просто ошеломляет и заставляет вспомнить не только о технических, но и о психологических методах профилактики подобных действий.

И, наконец, что же именно предпринимают злоумышленники, добравшись до информации: в 44% случаев взлома были произведены непосредственные кражи денег с электронных счетов, в 16% случаев выводилось из строя программное обеспечение, столь же часто – в 16% случаев – производилась кража информации с различными последствиями, в 12% случаев информация была сфальсифицирована, в 10% случаев злоумышленники с помощью компьютера воспользовались либо заказали услуги, к которым в принципе не должны были иметь доступа.

Для решения указанных и других подобных проблем не существует какого-либо одного технического приема или средства. Однако, общим в решении многих из них является использование криптографии и криптоподобных преобразований системы.

В основе криптографических методах лежит понятие криптографического преобразования информации, производимого по определенным математическим законам, с целью исключить доступ к данной информации

посторонних пользователей, а также с целью обеспечения невозможности бесконтрольного изменения информации со стороны тех же самых лиц.

Современные криптографические алгоритмы должны обладать следующими условиями:

- должны быть адаптированы к новейшей программно-аппаратной базе (например, алгоритмы блочного шифрования в программной реализации должны быть адаптированы к операциям с 64-разрядными числами);
- объем ключа должен соответствовать современным методам и средствам расшифрования зашифрованных сообщений;
- операции расшифрования и зашифрования должны по возможности быть простыми, чтобы удовлетворять современным требованиям по скоростным характеристикам;
- не должны допускать появления постоянно увеличивающегося числа ошибок;
- должны сводить к минимуму объем сообщения в ходе выполнения операции шифрования.

Основными функциями, возлагаемыми на аппаратную часть программного комплекса криптографической защиты информации являются генерация ключевой информации и хранение ключевой информации в устройствах, защищенных от несанкционированного доступа со стороны злоумышленника. Кроме того, посредством методик такого типа можно осуществлять аутентификацию пользователей с помощью паролей (статических или динамически изменяемых, хранящихся на переносимых устройствах).

Важное место играет длина и сложность ключа шифрования. Известно, чем длиннее ключ, тем он менее подвержен взлому, даже если определен алгоритм шифрования. Однако в умелых руках знающих программистов и хакеров, которые могут использовать в своем арсенале высоко сложные микропроцессорные системы последних разработок зашифрованный файл когда-нибудь может быть взломан и прочитан. Поэтому акцент надо делать на то, что содержимое файла очень скоро должно устареть и потерять свою

актуальность прежде, чем данный файл был взломан. В этом и есть главная защита любого типа данных – подверженность времени.

Необходимо указать цели и задачи данной работы.

Цель работы: разработка, исследование и практическая реализация модифицированного алгоритма шифрования данных RC6.

Задачи:

1. Исследовать основные теоретические положения проектирования и разработки алгоритмов шифрования.

2. Проанализировать и исследовать эффективность криптостойкости модифицированного алгоритма шифрования данных RC6.

3. Реализовать и исследовать модифицированный алгоритм RC6 с использованием инструментальных средств разработки Delphi.

Структура работы:

В первой главе приведена характеристика основных теоретических положений и выполнен анализ современных алгоритмов шифрования.

Во второй главе рассмотрены основные процедуры шифрования, расшифрования и генерации ключа стандартного и модифицированного алгоритмов RC6.

Третья глава содержит исследования по криптостойкости модифицированного RC6 с двумя вариантами шифрования, расшифрования и генерации ключа в своем практическом выполнении.

Приложение. Содержит подробное описание и скриншоты разработанной программы FileCoder.

При написании данной работы были использованы труды таких авторов, как С.Г.Баричев «Криптография без секретов»; С.Г.Баричев, Р.Е.Серов «Основы современной криптографии»; В.В.Ященко «Введение в криптографию»; О.В.Казарин «Безопасность программного обеспечения компьютерных систем» и многих других, чье изложение и подача материала послужило при создании первой теоретической главы данной работы.

Особое внимание хотелось бы уделить Г.Н.Устинову, который оказал неоценимую помощь при написании данной работы. Его работа «Основы

информационной безопасности» содержит обширный материал на самую разнообразную тематику, такую как организационные, правовые, экономические, юридические и социальные аспекты защиты информации в современном обществе.

Неоценимый вклад в развитии отечественной криптографии внес А.Винокуров. Его статья «Алгоритм шифрования ГОСТ 28147-89, его использование и реализация для компьютеров платформы Intel x86» содержит описание алгоритма, принятого в качестве стандарта шифрования в Российской Федерации, и его надежность реализации для процессоров семейства Intel x86, а также обсуждение различных вопросов его практического использования.

Также необходимо отметить А.В.Беляева и его работу «Методы и средства защиты информации», которая описывает многие современные алгоритмы шифрования данных и их схемы построения, одним из которых является алгоритм RC6. Его теоретические стандарты послужили основой при проектировании и разработке модифицированного алгоритма RC6.

Итак, в настоящее время криптографическая защита информации представляет собой актуальность и важность проблемы обеспечения безопасности информационных технологий.

1. КРАТКИЙ ОБЗОР И АНАЛИЗ СОВРЕМЕННЫХ АЛГОРИТМОВ ШИФРОВАНИЯ

Криптография – это наука об обеспечении безопасности данных. Она занимается поисками решений четырех важных проблем безопасности – конфиденциальности, аутентификации, целостности и контроля участников взаимодействия. Шифрование – это преобразование данных в нечитабельную форму, используя ключи шифрования/расшифрования. Шифрование позволяет обеспечить конфиденциальность, сохраняя информацию в тайне от того, кому она не предназначена.

1.1 Классификация криптоалгоритмов

Все современные криптографические алгоритмы используют схему шифрования, в которой в процессе шифрования/расшифрования помимо исходного текста используется секретный элемент – ключ, которым обладают только отправитель и получатель.

Укрупнено все криптосистемы подразделяются на симметричные и асимметричные [3. С. 18].

1. Симметричные криптосистемы (с секретным ключом – secret key systems*)* – данные криптосистемы построены на основе сохранения в тайне ключа шифрования. Процессы шифрования и расшифрования используют один и тот же ключ. Секретность ключа является постулатом. Основная проблема при применении симметричных криптосистем для связи заключается в сложности передачи обоим сторонам секретного ключа. Однако данные системы обладают высоким быстродействием. Раскрытие ключа злоумышленником грозит раскрытием только той информации, что была зашифрована на этом ключе. Американский и Российский стандарты шифрования DES, AES и ГОСТ 28147-89 – все эти алгоритмы являются представителями симметричных криптосистем. В табл. 1 представлены наиболее популярные симметричные алгоритмы.

Табл. 1. Популярные симметричные криптоалгоритмы

Тип	Описание
DES (Data Encryption Standard)	Популярный алгоритм шифрования, используемый как стандарт шифрования данных правительством США. Шифруется блок из 64 бит, используется 64-битовый ключ (требуется только 56 бит), 16 проходов Может работать в 4 режимах: • Электронная кодовая книга (ECB-Electronic Code Book) - обычный DES, использует два различных алгоритма. • Цепочечный режим (CBC-Cipher Block Chaining), в котором шифрование шифрование блока данных зависит от результатов шифрования предыдущих блоков данных. • Обратная связь по выходу (OFB-Output Feedback), используется как генератор случайных чисел. • Обратная связь по шифратору (CFB-Cipher Feedback), используется для получения кодов аутентификации сообщений.
3-DES или тройной DES	64-битный блочный шифратор, использует DES 3 раза с тремя различными 56-битными ключами. Достаточно стоек ко всем атакам
Каскадный 3-DES	Стандартный тройной DES, к которому добавлен механизм обратной связи, такой как CBC, OFB или CFB Очень стоек ко всем атакам.
FEAL (быстрый алгоритм шифрования)	Блочный шифратор, используемый как альтернатива DES Вскрыт, хотя после этого были предложены новые версии.
IDEA (международный алгоритм шифрования)	64-битный блочный шифратор, 128-битовый ключ, 8 проходов Предложен недавно; хотя до сих пор не прошел полной проверки, чтобы считаться надежным, считается более лучшим, чем DES
Skipjack	Разработано АНБ в ходе проектов правительства США "Clipper" и "Capstone". До недавнего времени был секретным, но его стойкость не зависела только от того, что он был секретным. 64-битный блочный шифратор, 80-битовые ключи используются в режимах ECB, CFB, OFB или CBC, 32 прохода
RC2	64-битный блочный шифратор, ключ переменного размера Приблизительно в 2 раза быстрее, чем DES Может использоваться в тех же режимах, что и DES, включая тройное шифрование. Конфиденциальный алгоритм, владельцем которого является RSA Data Security
RC4	Потоковый шифр, байт-ориентированный, с ключом переменного размера. Приблизительно в 10 раз быстрее DES. Конфиденциальный алгоритм, которым владеет RSA Data Security
RC5	Имеет размер блока 32, 64 или 128 бит, ключ с длиной от 0 до 2048 бит, от 0 до 255 проходов

	Быстрый блочный шифр Алгоритм, которым владеет RSA Data Security
CAST	64-битный блочный шифратор, ключи длиной от 40 до 64 бит, 8 проходов Неизвестно способов вскрыть его иначе как путем прямого перебора.
Blowfish.	64-битный блочный шифратор, ключ переменного размера до 448 бит, 16 проходов, на каждом проходе выполняются перестановки, зависящие от ключа, и подстановки, зависящие от ключа и данных. Быстрее, чем DES Разработан для 32-битных машин
Устройство с одноразовыми ключами	Шифратор, который нельзя вскрыть. Ключом (который имеет ту же длину, что и шифруемые данные) являются следующие 'n' бит из массива случайно созданных бит, хранящихся в этом устройстве. У отправителя и получателя имеются одинаковые устройства. После использования биты разрушаются, и в следующий раз используются другие биты.
Поточные шифры	Быстрые алгоритмы симметричного шифрования, обычно оперирующие битами (а не блоками бит). Разработаны как аналог устройства с одноразовыми ключами, и хотя не являются такими же безопасными, как оно, по крайней мере практичны.

2. Асимметричные криптосистемы (системы открытого шифрования, с открытым ключом – public key systems) – смысл данных криптосистем состоит в том, что для шифрования и расшифрования используются разные преобразования [3. С.25].

Суть их состоит в том, что каждым адресатом информационной системы генерируются два ключа, связанные между собой по определенному правилу. Один ключ объявляется открытым, а другой закрытым. Открытый ключ публикуется и доступен любому, кто желает послать сообщение адресату. Секретный ключ сохраняется в тайне.

Исходный текст шифруется открытым ключом адресата и передается ему. Зашифрованный текст в принципе не может быть расшифрован тем же открытым ключом. Расшифрование сообщения возможно только с использованием закрытого ключа, который известен только самому адресату (рис. 1).

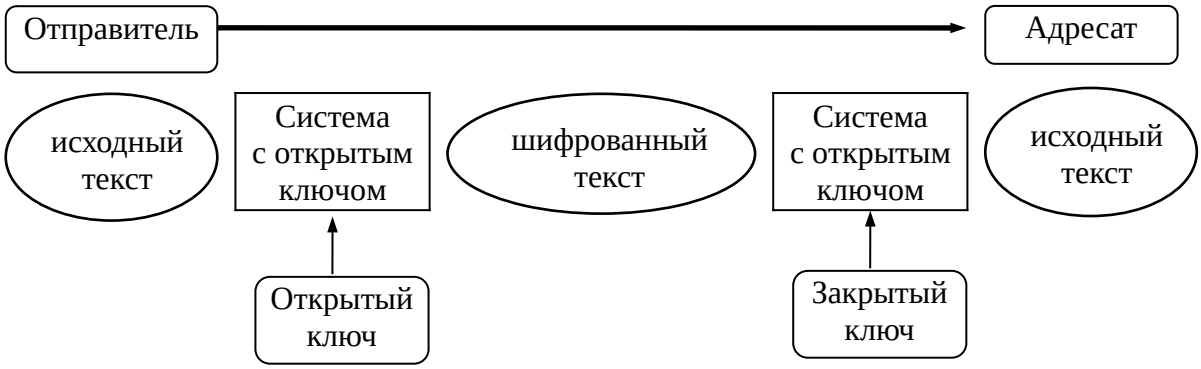


Рис. 1. Асимметричный алгоритм

Таким образом, пропадает проблема передачи секретного ключа (как у симметричных систем). Однако, несмотря на все свои преимущества, эти криптосистемы достаточно трудоемки и медлительны. Стойкость асимметричных криптосистем базируется, в основном, на алгоритмической трудности решить за приемлемое время какую-либо задачу. Если злоумышленнику удастся построить такой алгоритм, то дискредитирована будет вся система и все сообщения, зашифрованные с помощью этой системы. В этом состоит главная опасность асимметричных криптосистем в отличие от симметричных.

Поэтому чтобы гарантировать надежную защиту информации, к системам с открытым ключом предъявляются два важных и очевидных требования [4. С. 32-37]:

1. Преобразование исходного текста должно быть необратимым и исключать его восстановление на основе открытого ключа.

2. Определение закрытого ключа на основе открытого также должно быть невозможным на современном технологическом уровне. При этом желательна точная нижняя оценка сложности (количества операций) раскрытия шифра.

Алгоритмы шифрования с открытым ключом получили широкое распространение в современных информационных системах. Примеры – системы открытого шифрования RSA, система Рабина, Эль Гамаля, система на основе эллиптических уравнений и другие. В табл. 2 представлены наиболее популярные асимметричные алгоритмы.

Табл. 2. Популярные асимметричные криптоалгоритмы

Тип	Описание
RSA	Популярный алгоритм асимметричного шифрования, стойкость которого зависит от сложности факторизации больших целых чисел.
ECC (криптосистема на основе эллиптических кривых)	Использует алгебраическую систему, которая описывается в терминах точек эллиптических кривых, для реализации асимметричного алгоритма шифрования. Является конкурентом по отношению к другим асимметричным алгоритмам шифрования, так как при эквивалентной стойкости использует ключи меньшей длины и имеет большую производительность. Современные его реализации показывают, что эта система гораздо более эффективна, чем другие системы с открытыми ключами. Его производительность приблизительно на порядок выше, чем производительность RSA, Диффи-Хеллмана и DSA.
Эль-Гамаль.	Вариант Диффи-Хеллмана, который может быть использован как для шифрования, так и для электронной подписи.

В зависимости от размера блока информации криптоалгоритмы делятся на [4. С. 45-60]:

1. Потоковые шифры. Единицей кодирования является один бит. Результат кодирования не зависит от прошедшего ранее входного потока. Схема применяется в системах передачи потоков информации, то есть в тех случаях, когда передача информации начинается и заканчивается в произвольные моменты времени и может случайно прерываться. Наиболее распространенными предстателями поточных шифров являются скремблеры [6. С. 55-60].

2. Шифры перестановок. Шифрование перестановкой заключается в том, что символы шифруемого текста переставляются по определенному правилу в пределах некоторого блока этого текста. При достаточной длине блока, в пределах которого осуществляется перестановка, и сложном неповторяющемся порядке перестановки можно достигнуть приемлемой для простых практических приложений стойкости шифра.

3. Шифрование заменой (подстановкой) заключается в том, что символы шифруемого текста заменяются символами того же или другого алфавита в соответствии с заранее обусловленной схемой замены.

4. Гаммирование. Гаммирование является также широко применяемым криптографическим преобразованием. Принцип шифрования гаммированием заключается в генерации гаммы шифра с помощью датчика псевдослучайных чисел и наложении полученной гаммы на открытые данные обратимым образом (например, используя сложение по модулю 2) [7. С. 10-15].

Процесс расшифрования данных сводится к повторной генерации гаммы шифра при известном ключе и наложении такой гаммы на зашифрованные данные.

Полученный зашифрованный текст является достаточно трудным для раскрытия в том случае, если гамма шифра не содержит повторяющихся битовых последовательностей. По сути дела гамма шифра должна изменяться случайным образом для каждого шифруемого слова. Фактически же, если период гаммы превышает длину всего зашифрованного текста и неизвестна никакая часть исходного текста, то шифр можно раскрыть только прямым перебором (пробой на ключ). Криптостойкость в этом случае определяется размером ключа.

Метод гаммирования становится бессильным, если злоумышленнику становится известен фрагмент исходного текста и соответствующая ему шифрограмма. Злоумышленники может сделать это на основе догадок о содержании исходного текста. Так, если большинство посылаемых сообщений начинается со слов “СОВ.СЕКРЕТНО”, то криптоанализ всего текста значительно облегчается. Это следует учитывать при создании реальных систем информационной безопасности.

5. Блочные шифры. Единицей кодирования является блок из нескольких байтов (в настоящее время 4-32). Результат кодирования зависит от всех исходных байтов этого блока. К блочным шифрам предъявляются следующие требования [5. С. 25]:

- достаточная криптостойкость;
- простота процедур зашифрования и расшифрования;
- приемлемая надежность;

Блочные алгоритмы шифрования являются основным средством криптографической защиты информации, хранящейся на компьютере пользователя или передаваемой по общедоступной сети. Они обладают следующими преимуществами практического применения:

- возможность эффективной программной реализации на современных аппаратно-программных средствах;
- высокую скорость зашифрования/расшифрования как при аппаратной, так и программной реализации;
- высокую гарантированную стойкость; причем стойкость алгоритма блочного шифрования может быть доказана при помощи математического аппарата (для большинства ассиметричных алгоритмов стойкость основана на «невозможности» решения какой-либо математической задачи).

Суть алгоритмов блочного шифрования заключается в применении к блоку открытого текста многократного математического преобразования. Многократность подобных операций приводит к тому, что результирующее преобразование оказывается криптографически более сильным, чем преобразование над отдельно взятым блоком. Основная цель подобного трансформирования – создать зависимость каждого бита ключа и каждого бита блока открытого сообщения.

Характерной особенностью блочных криптоалгоритмов является тот факт, что в ходе своей работы они производят преобразование блока входной информации фиксированной длины и получают результирующий блок того же объема, но недоступный для прочтения сторонним лицам, не владеющим ключом. Таким образом, схему работы блочного шифра можно описать функциями Z=EnCrypt(X,Key) и X=DeCrypt(Z,Key) [11. С. 34-45].

Ключ Key является параметром блочного криптоалгоритма и представляет собой некоторый блок двоичной информации фиксированного размера. Исходный (X) и зашифрованный (Z) блоки данных также имеют фиксированную разрядность, равную между собой, но необязательно равную длине ключа.

Блочные шифры являются основой, на которой реализованы практически все криптосистемы. Методика создания цепочек из зашифрованных блочными алгоритмами байт позволяет шифровать ими пакеты информации неограниченной длины. Такое свойство блочных шифров, как быстрота работы, используется асимметричными криптоалгоритмами, медлительными по своей природе. Отсутствие статистической корреляции между битами выходного потока блочного шифра используется для вычисления контрольных сумм пакетов данных и в хешировании паролей.

Таким образом, на функцию стойкого блочного шифра Z=EnCrypt(X,Key) накладываются следующие условия:

1. Функция EnCrypt должна быть обратимой.

2. Не должно существовать иных методов прочтения сообщения X по известному блоку Z, кроме как полным перебором ключей Key.

3. Не должно существовать иных методов определения каким ключом Key было произведено преобразование известного сообщения X в сообщение Z, кроме как полным перебором ключей.

Все действия, производимые над данными блочным криптоалгоритмом, основаны на том факте, что преобразуемый блок может быть представлен в виде целого неотрицательного числа из диапазона, соответствующего его разрядности. Так, например, 32-битный блок данных можно интерпретировать как число из диапазона 0..4 294 967 295. Кроме того, блок, разрядность которого обычно является «степенью двойки», можно трактовать как несколько независимых неотрицательных чисел из меньшего диапазона (рассмотренный выше 32-битный блок можно также представить в виде 2 независимых чисел из диапазона 0..65535 или в виде 4 независимых чисел из диапазона 0..255) [12 С. 12].

Характерным признаком блочных алгоритмов является многократное и косвенное использование материала ключа. Это диктуется в первую очередь требованием невозможности обратного декодирования в отношении ключа при известных исходном и зашифрованном текстах. Для решения этой задачи в приведенных выше преобразованиях чаще всего используется не само значение

ключа или его части, а некоторая, иногда необратимая (небиективная) функция от материала ключа. Более того, в подобных преобразованиях один и тот же блок или элемент ключа используется многократно. Это позволяет при выполнении условия обратимости функции относительно величины X сделать функцию необратимой относительно ключа Key.

Поскольку операция шифрования или расшифрования отдельного блока в процессе кодирования пакета информации выполняется многократно (иногда до сотен тысяч раз), а значение ключа и, следовательно, функций $V_i(Key)$ остается неизменным, то иногда становится целесообразно заранее однократно вычислить данные значения и хранить их в оперативной памяти совместно с ключом. Поскольку эти значения зависят только от ключа, то они в криптографии называются материалом ключа. Необходимо отметить, что данная операция никоим образом не изменяет ни длину ключа, ни криптостойкость алгоритма в целом.

1.2 Сеть Фейштеля как популярный атрибут блочных шифров

Сеть Фейштеля является методом смешивания текущей части шифруемого блока с результатом некоторой функции, вычисленной от другой независимой части того же блока. Эта методика получила широкое распространение, поскольку обеспечивает выполнение требования о многократном использовании ключа и материала исходного блока информации [5. С. 30-40].

Классическая сеть Фейштеля имеет следующую структуру (рис. 2):

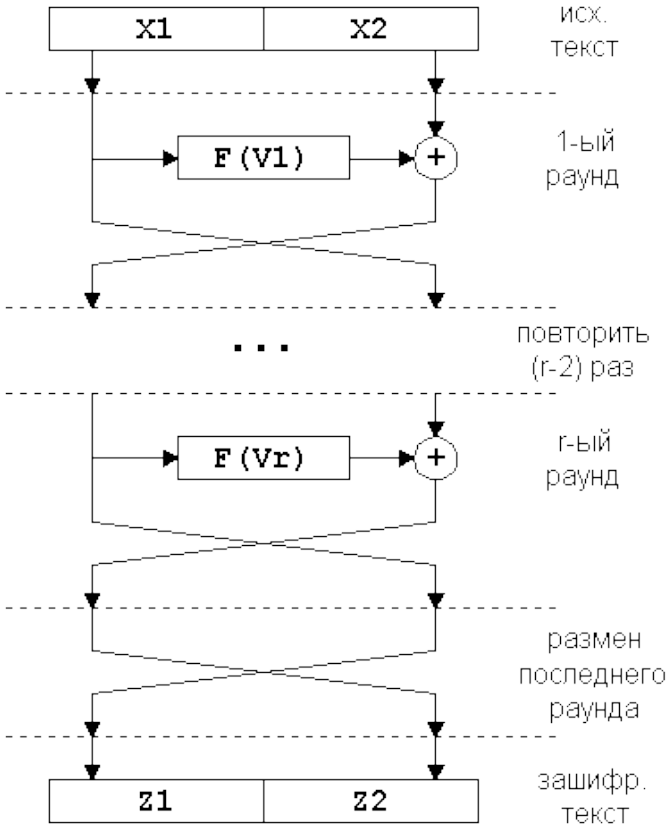


Рис. 2. Сеть Фейштеля

Независимые потоки информации, порожденные из исходного блока, называются ветвями сети. В классической схеме их две. Величины V_i именуются параметрами сети, обычно это функции от материала ключа. Функция F называется образующей. Действие, состоящее из однократного вычисления образующей функции и последующего наложения ее результата на другую ветвь с обменом их местами, называется циклом или раундом (англ. round) сети Фейштеля. Оптимальное число раундов K – от 8 до 32. Важно то, что увеличение количества раундов значительно увеличивает криптоскойстость любого блочного шифра к криптоанализу. Возможно, эта особенность и повлияла на столь активное распространение сети Фейштеля – ведь при обнаружении, скажем, какого-либо слабого места в алгоритме, почти всегда достаточно увеличить количество раундов на 4-8, не переписывая сам алгоритм. Часто количество раундов не фиксируется разработчиками алгоритма, а лишь указываются разумные пределы (обязательно нижний, и не всегда – верхний) этого параметра.

Сразу же возникает вопрос, – является ли данная схема обратимой? Очевидно, да. Сеть Фейштеля обладает тем свойством, что даже если в качестве образующей функции F будет использовано необратимое преобразование, то и в этом случае вся цепочка будет восстановима. Это происходит вследствие того, что для обратного преобразования сети Фейштеля не нужно вычислять функцию F^{-1}.

Более того, как нетрудно заметить, сеть Фейштеля симметрична. Использование операции XOR, обратимой своим же повтором, и инверсия последнего обмена ветвей делают возможным раскодирование блока той же сетью Фейштеля, но с инверсным порядком параметров V_i. Заметим, что для обратимости сети Фейштеля не имеет значение является ли число раундов четным или нечетным числом. В большинстве реализаций схемы, в которых оба вышеперечисленные условия (операция XOR и уничтожение последнего обмена) сохранены, прямое и обратное преобразования производятся одной и той же процедурой, которой в качестве параметра передается вектор величин V_i либо в исходном, либо в инверсном порядке.

С незначительными доработками сеть Фейштеля можно сделать и абсолютно симметричной, то есть выполняющей функции шифрования и расшифрования одним и тем же набором операций. Математическим языком это записывается как «Функция EnCrypt тождественно равна функции DeCrypt». Если мы рассмотрим граф состояний криптоалгоритма, на котором точками отмечены блоки входной и выходной информации, то при каком-то фиксированном ключе для классической сети Фейштеля мы будем иметь картину, изображенную на рис. 3а, а во втором случае каждая пара точек получит уникальную связь, как изображено на рис. 3б.

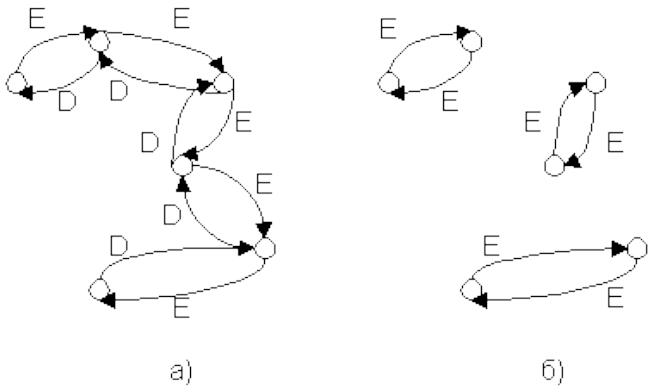


Рис. 3

Модификация сети Фейштеля, обладающая подобными свойствами приведена на рис. 4. Как видим, основная ее хитрость в повторном использовании данных ключа в обратном порядке во второй половине цикла. Необходимо заметить, однако, что именно из-за этой недостаточно исследованной специфики такой схемы (то есть потенциальной возможности ослабления зашифрованного текста обратными преобразованиями) ее используют в криптоалгоритмах с большой осторожностью.

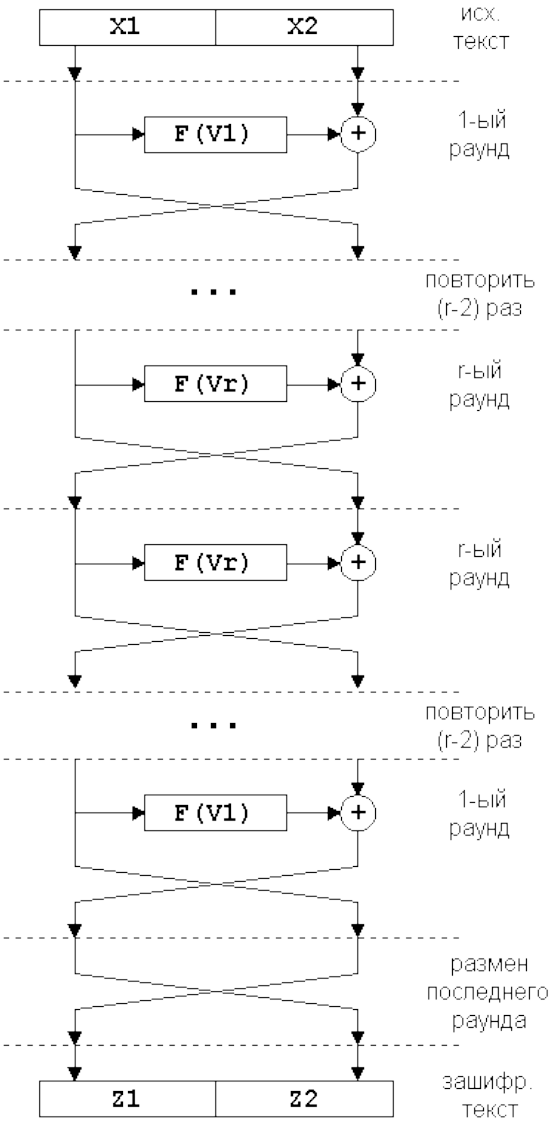


Рис. 4. Модификация сети Фейштеля

А вот модификацию сети Фейштеля для большего числа ветвей применяют гораздо чаще. Это в первую очередь связано с тем, что при больших размерах кодируемых блоков (128 и более бит) становится неудобно работать с математическими функциями по модулю 64 и выше. Как известно, основные единицы информации обрабатываемые процессорами на сегодняшний день – это байт и двойное машинное слово 32 бита. Поэтому все чаще и чаще в блочных криптоалгоритмах встречается сеть Фейштеля с 4-мя ветвями. Самый простой принцип ее модификации изображен на рис. 5а. Для более быстрого перемешивания информации между ветвями (а это основная проблема сети Фейштеля с большим количеством ветвей) применяются две

модифицированные схемы, называемые «type-2» и «type-3». Они изображены на рис. 5б и 5в.

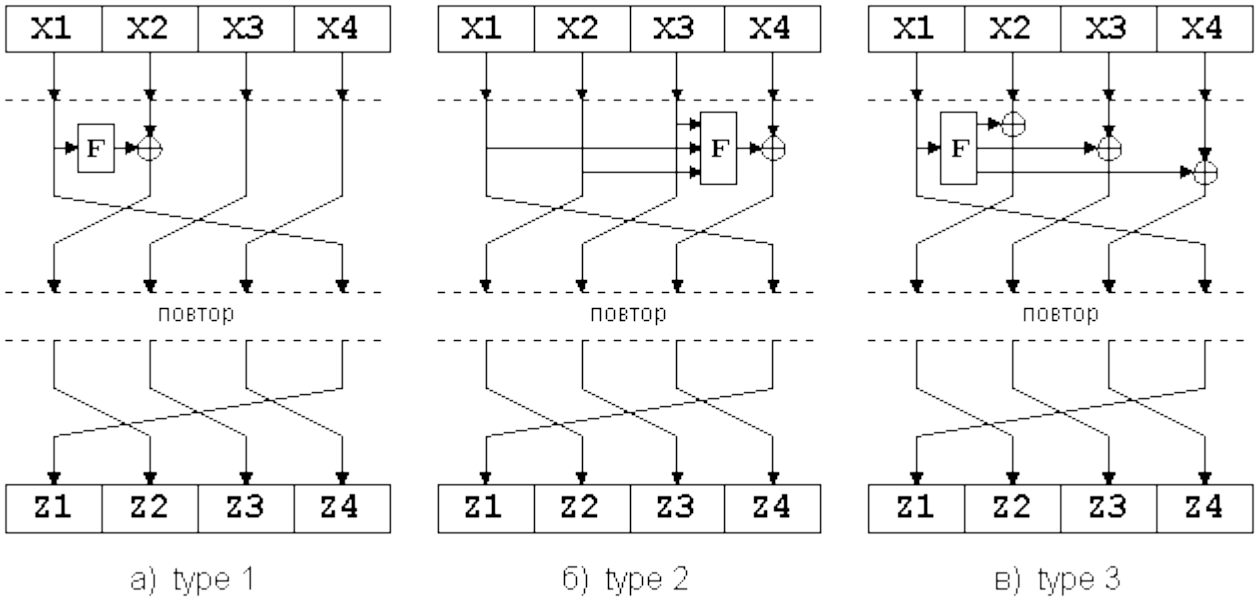


Рис. 5. Модификация сети Фейштеля для большего числа ветвей

Сеть Фейштеля надежно зарекомендовала себя как криптостойкая схема произведения преобразований, и ее можно найти практически в любом современном блочном шифре. Незначительные модификации касаются обычно дополнительных начальных и оконечных преобразований (англоязычный термин – whitening) над шифруемым блоком. Подобные преобразования, выполняемые обычно также либо «исключающим ИЛИ» или сложением имеют целью повысить начальную рандомизацию входного текста. Таким образом, криптостойкость блочного шифра, использующего сеть Фейштеля, определяется на 95% функцией F и правилом вычисления V_i из ключа. Эти функции и являются объектом все новых и новых исследований специалистов в области криптографии.

1.3 Стандарт шифрования данных DES и AES

Одним из самых распространенных алгоритмов блочного шифрования, рекомендованных Национальным бюро стандартов США в качестве основного

средства криптографической защиты информации как в государственных, так и в коммерческих структурах, является DES (Data Encryption Standard), который получил достаточно широкое распространение в свое время. Он был разработан в 1977г., однако уже в 1988г [3. С. 34]. Национальное бюро стандартов рекомендовало использовать DES только в системах электронного перевода. С учетом выявленных недостатков DES в начальный вариант стандарта постоянно вносятся изменения; появляются также и новые алгоритмы, использующие в качестве основы DES – NewDES, TripleDES, LUCIFER, MADRYGA, FEAL, REDOC, LOKI, KHUFU, KHAFRE, IDEA, MMB, CA-1.1, SKIPJACK, CAST, BLOWFISH, SAFER, 3-WAY, CRAB, и другие. Однако на текущий момент стандарт DES полностью неприемлем для использования по двум причинам:

1. Основной – длина его ключа составляет 56 бит, что чрезвычайно мало на современном этапе развития ЭВМ.

2. Второстепенной – при разработке алгоритм был ориентирован на аппаратную реализацию, то есть содержал операции, выполняемые на микропроцессорах за неприемлемо большое время (например, такие как перестановка бит внутри машинного слова по определенной схеме).

Все это сподвигло Американский институт стандартизации NIST (National Institute of Standards & Technology) в 1996 г на поиски нового стандарта симметричного криптоалгоритма, впоследствии названного AES (Advanced Encryption Standard), который стал де-факто мировым криптостандартом на ближайшие 10-20 лет. На сей раз уже были учтены основные промахи шифра-предшественника, а к разработке были подключены самые крупные центры по криптологии со всего мира.

Требования, предъявленные на AES, были предельно просты [5. С. 51]:

1. Алгоритм должен быть симметричным.

2. Алгоритм должен быть блочным шифром.

3. Алгоритм должен иметь длину блока 128 бит, и поддерживать три длины ключа: 128, 192 и 256 бит.

В течение 2 лет специалисты комитета, исследуя самостоятельно, и изучая публикации других исследователей, выбрали 5 лучших представителей (Табл. 3).

Табл. 3. 5 лучших представителей AES

Алгоритм	**Создатель**	**Страна**	**Быстродействие (asm, 200МГц)**
MARS	IBM	US	8 Мбайт/с
RC6	R.Rivest & Co	US	12 Мбайт/с
Rijndael	V.Rijmen & J.Daemen	BE	7 Мбайт/с
Serpent	Universities	IS, UK, NO	2 Мбайт/с
TwoFish	B.Schneier & Co	US	11 Мбайт/с

Все эти алгоритмы были признаны достаточно стойкими и успешно противостоящими всем широко известным методам криптоанализа.

2 октября 2000 года NIST объявил, что самым лучшим криптоалгоритмом шифрования данных является алгоритм Rijndael (произносится «Рейн-дал»). Этот алгоритм был разработан Винсентом Райманом и Йоан Даменом и представляет собой алгоритм, не использующий сети Фейштеля. С этого момента с алгоритма-победителя сняты все патентные ограничения – его можно будет использовать в любой криптопрограмме без отчисления каких-либо средств создателю.

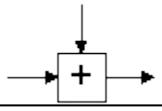
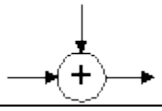
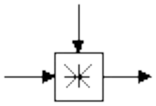
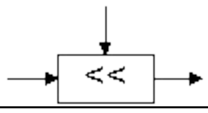
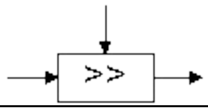
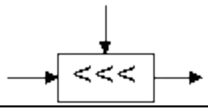
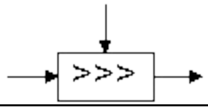
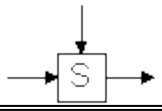
1.4 Схемы построения широко распространенных алгоритмов шифрования

В данном разделе описываются симметричные криптосистемы AES и отечественный стандарт ГОСТ 28147-89, а также наиболее популярные асимметричные алгоритмы шифрования данных RSA, криптосистема Эль-Гамаля.

1.4.1 Обзор симметричных криптосистем

Все нижеследующие симметричные блочные криптоалгоритмы содержат следующие обозначения и производятся по определенной схеме (слева даны условные обозначения этих операций на графических схемах алгоритмов) (Табл. 4):

Табл. 4. Обозначения операций преобразования

Биективные математические функции		
	Сложение	X'=X+V
	Исключающее ИЛИ	X'=X XOR V
	Умножение по модулю 2^N+1	X'=(X*V) mod (2^N+1)
	Умножение по модулю 2^N	X'=(X*V) mod (2^N)
Битовые сдвиги		
	Арифметический сдвиг влево	X'=X SHL V
	Арифметический сдвиг вправо	X'=X SHR V
	Циклический сдвиг влево	X'=X ROL V
	Циклический сдвиг вправо	X'=X ROR V
Табличные подстановки		
	S-box (англ. substitute)	X'=Table[X,V]

В качестве параметра V для любого из этих преобразований может использоваться [5. С. 77-98]:

1. Фиксированное число (например, X'=X+125)
2. Число, получаемое из ключа (например, X'=X+F(Key))
3. Число, получаемое из независимой части блока (например, $X_2'=X_2+F(X_1)$)

1.4.1.1 Шифр MARS

Шифр состоит из трех видов операций, которые повторяются сначала в прямом, а затем в инверсном порядке. На первом шаге идет классическое входное забеливание: ко всем байтам исходного текста добавляются байты из материала ключа.

Второй этап: прямое перемешивание, однотипная операция, имеющая структуру сети Фейштеля повторяется 8 раз. Однако, на этом этапе не производится добавление материала ключа. Цель данного преобразования – тщательная рандомизация данных и повышение стойкости шифра к некоторым видам атак (рис.6).

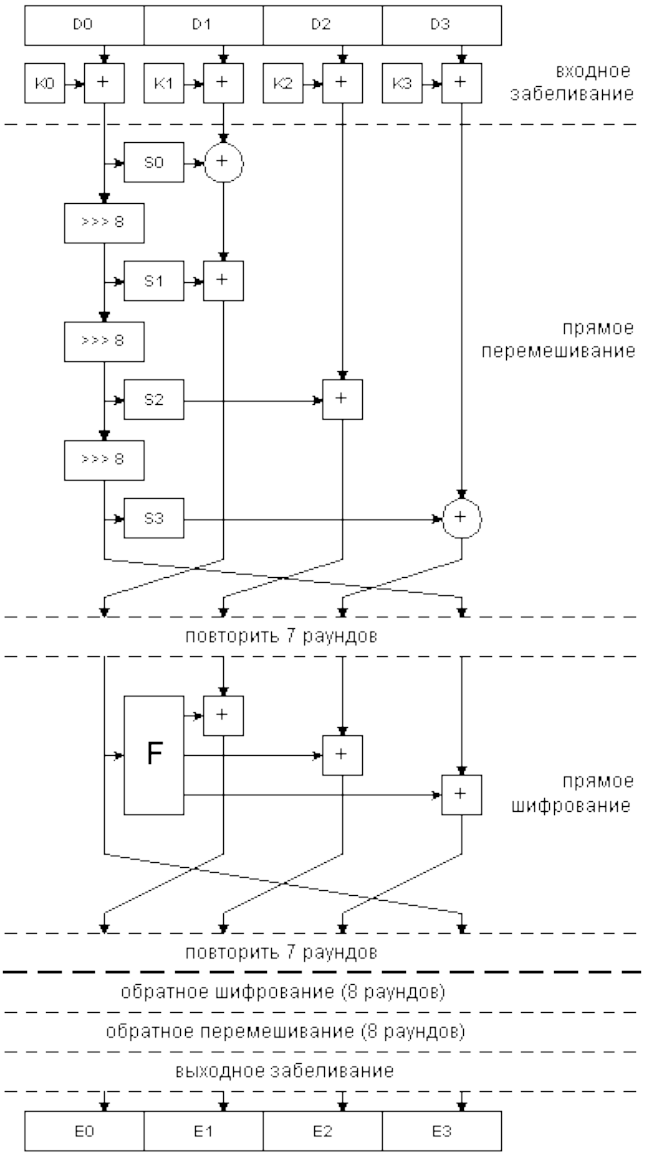


Рис. 6. Шифр MARS

Третий этап: собственно шифрование. В нем используется сеть Фейштеля треьего типа с 4 ветвями, то есть значения трех функций, вычисленных от одной ветви накладываются соответственно на три других, затем идет перестановка машинных слов. Эта операция также повторяется 8 раз (рис.6). Именно на этом этапе происходит смешивание текста с основной (большей) частью материала ключа. Сами функции, накладываемые на ветви, изображены на рис. 7. Как видим, в алгоритме MARS использованы практически все виды операций, применяемых в криптографических преобразованиях: сложение, «исключающее ИЛИ», сдвиг на фиксированное число бит, сдвиг на переменное число бит, умножение и табличные подстановки.

Во второй части операции шифрования повторяются те же операции, но в обратном порядке: сначала шифрование, затем перемешивание, и, наконец, забеливание. При этом во вторые варианты всех операций внесены некоторые изменения таким образом, чтобы криптоалгоритм в целом стал абсолютно симметричным. То есть, в алгоритме MARS для любого X выполняется выражение EnCrypt(EnCrypt(X))=X.

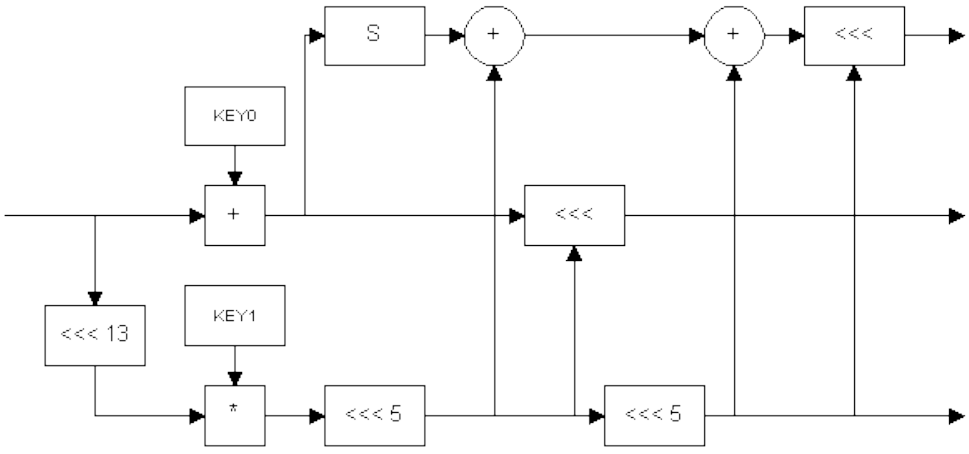


Рис. 7. Функции шифра MARS

1.4.1.2 Шифр Serpent

Алгоритм разработан группой ученых из нескольких исследовательских центров мира. Алгоритм представляет собой сетей Фейштеля для четырех ветвей смешанного типа (рис 8) [3. С. 45]: 2 четные ветви изменяют совместно значения нечетных, затем меняются местами. В качестве криптопреобразований используются только «исключающее ИЛИ», табличные подстановки и битовые сдвиги. Алгоритм состоит из 32 раундов. Сами раунды составлены таким образом, что добавление к ветвями материала ключа на первом и последнем раундах образует входное и выходное забеливание.

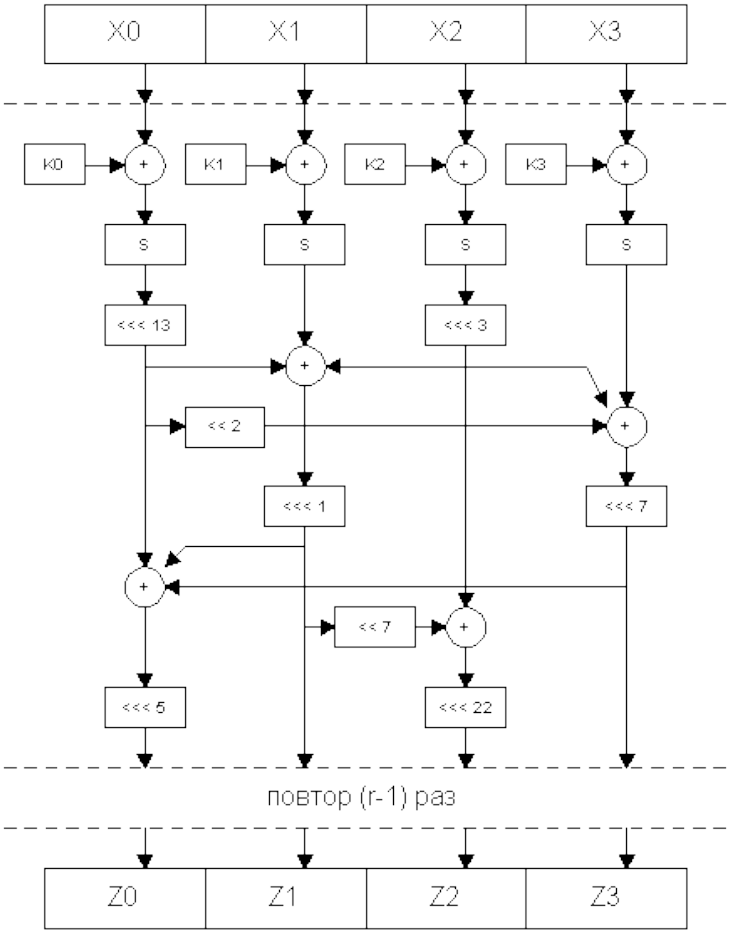


Рис. 8. Шифр Serpent

1.4.1.3 Шифр TwoFish

Алгоритм разработан комманией Counterpain Security Systems, возглавляемой Брюсом Шнайером (англ. Bruce Schneier). Предыдущая

программная разработка этой фирмы, называвшаяся BlowFish, являлась и до сих пор является признанным криптостойким алгоритмом [4. С. 78].

В алгоритме TwoFish разработчики оставили некоторые удачные решения из проекта-предшественника, кроме этого произвели тщательные исследования по перемешиванию данных в сети Фейштеля. Алгоритм представляет собой сеть Фейштеля смешанного типа: первая и вторая ветвь на нечетных раундах производят модификацию третьей и четвертой, на четных раундах ситуация меняется на противоположную. В алгоритме используется криптопреобразование Адамара (англ. Pseudo-Hadamar Transform) – обратимое арифметическое сложение первого потока со вторым, а затем второго с первым.

Единственным нарицанием, поступившим в адрес TwoFish от независимых исследователей, является тот факт, что при расширении материала ключа в алгоритме используется сам же алгоритм. Двойное применение блочного шифра довольно сильно усложняет его анализ на предмет наличия слабых ключей или недокументированных замаскированных связей между входными и выходными данными (рис. 9).

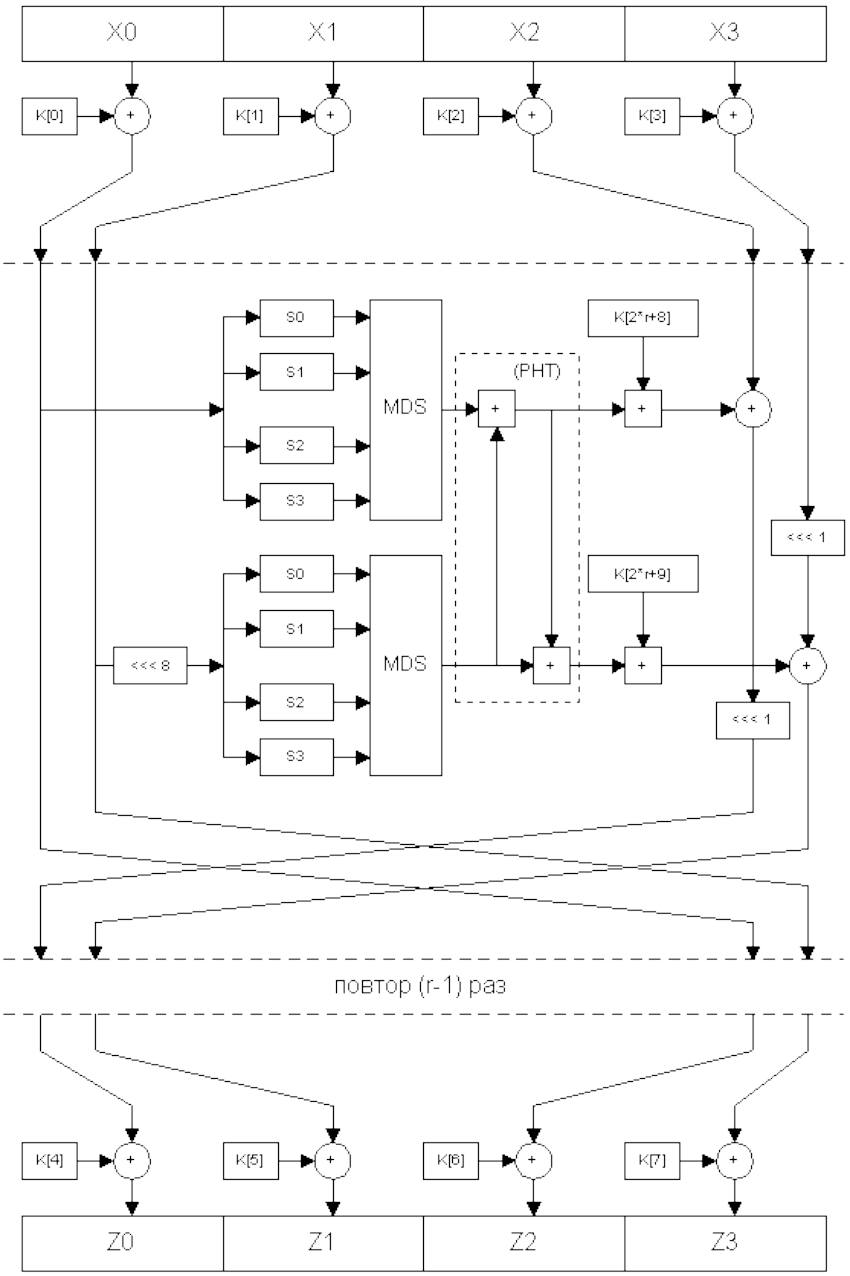


Рис. 9. Шифр TwoFish

1.4.1.4 Шифр Rijndael

Данный алгоритм разработан двумя специалистами по криптографии из Бельгии. Он является нетрадиционным блочным шифром, поскольку не использует сеть Фейштеля для криптопреобразований. Алгоритм представляет каждый блок кодируемых данных в виде двумерного массива байт размером 4x4, 4x6 или 4x8 в зависимости от установленной длины блока. Далее на соответствующих этапах преобразования производятся либо над независимыми

столбцами, либо над независимыми строками, либо вообще над отдельными байтами в таблице [11. С. 66-77].

Все преобразования в шифре имеют строгое математическое обоснование. Сама структура и последовательность операций позволяют выполнять данный алгоритм эффективно как на 8-битных так и на 32-битных процессорах. В структуре алгоритма заложена возможность параллельного исполнения некоторых операций, что на многопроцессорных рабочих станциях может еще поднять скорость шифрования в 4 раза.

Алгоритм состоит из некоторого количества раундов (от 10 до 14 – это зависит от размера блока и длины ключа), в которых последовательно выполняются следующие операции :

ByteSub – табличная подстановка 8x8 бит (рис. 10),

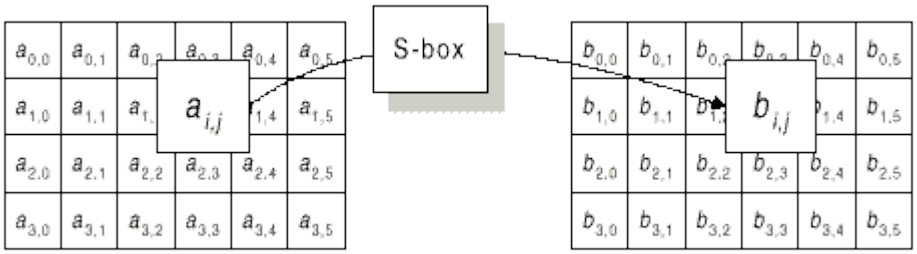


Рис. 10

ShiftRow – сдвиг строк в двумерном массиве на различные смещения (рис.11),

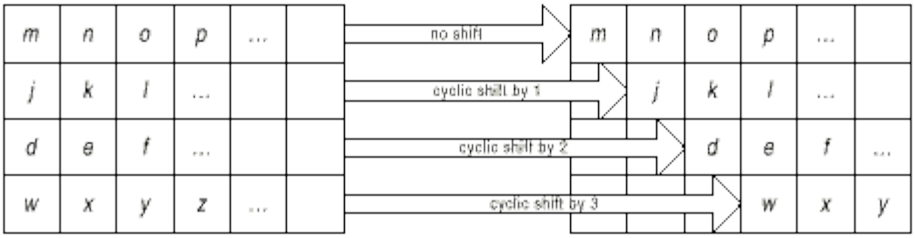


Рис. 11

MixColumn – математическое преобразование, перемешивающее данные внутри столбца (рис.12),

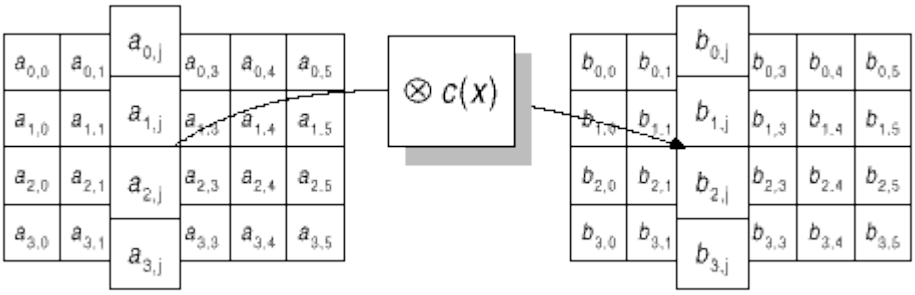


Рис. 12

AddRoundKey – добавление материала ключа операцией XOR (рис.13).

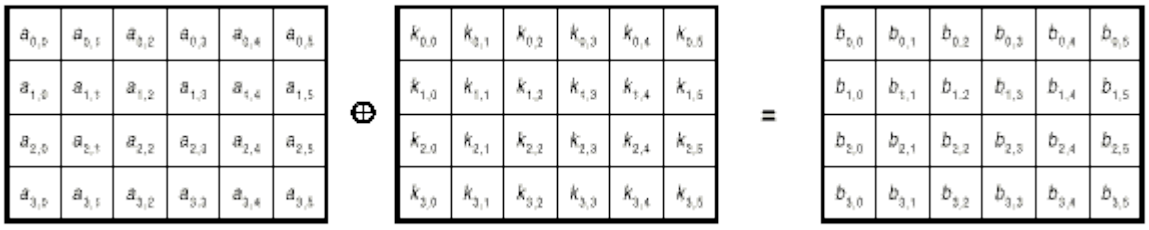


Рис. 13

В последнем раунде операция перемешивания столбцов отсутствует, что делает всю последовательность операций симметричной.

1.4.1.5 Стандарт шифрования данных ГОСТ 28147-89

Также следует указать российский стандарт криптографического преобразования текста для информационных систем. Он носит обязательный характер для государственных органов, организаций, предприятий, банковских и иных учреждений, чья деятельность связана с обеспечением информационной безопасности государства. Для других организаций и частных лиц, ГОСТ имеет рекомендательный характер [8].

Данный стандарт формировался с учетом мирового опыты и, в частности, были приняты во внимание недостатки и нереализованные возможности алгоритма DES, поэтому использование стандарта ГОСТ предпочтительнее. Алгоритм шифрования построен с использованием сети Фейштеля. Алгоритм достаточно сложен и ниже будет описана в основном его концепция.

Будем использовать следующие операции сложения:

- A⊕B - побитовое сложение по модулю 2;
- A[+]B - сложение по модулю 2^{32};
- A{+}B - сложение по модулю 2^{32}-1;

Алгоритм криптографического преобразования предусматривает несколько режимов работы: 1) простая замена; 2) гаммирование; 3) гаммирование с обратной связью. Во всех режимах используется ключ W длиной 256 бит, представляемый в виде восьми 32-разрядных чисел x(i).

W=X(7)X(6)X(5)X(4)X(3)X(2)X(1)X(0)

Для расшифрования используется тот же ключ, но процесс расшифрования является инверсным по отношению к исходному. Самый простой из возможных режимов – простая замена.

1. Простая замена. Пусть открытые блоки разбиты на блоки по 64 бит в каждом, которые обозначим как T(j). Очередная последовательность бит T(j) разделяется на две последовательности B(0) и A(0) по 32 бита (правый и левый блоки). Далее выполняется итеративный процесс шифрования описываемый следующими формулами, вид который зависит от i:

- Для i=1, 2, ..., 24, j=(i-1) mod 8;

$$A(i) = f(A(i-1)\ [+]\ x(j)) \oplus B(i-1)$$

$$B(i) = A(i-1)$$

- Для i=25, 26, ..., 31, j=32-i;

$$A(i) = f(A(i-1)\ [+]\ x(j)) \oplus B(i-1)$$

$$B(i) = A(i-1)$$

- Для i=32

$$A(32) = A(31)$$

$$B(32) = f(A(31)\ [+]\ x(0)) \oplus B(31).$$

Здесь i обозначает номер итерации. Функция f – функция шифрования.

Функция шифрования включает две операции над 32-разрядным аргументом.

Первая операция является подстановкой K. Блок подстановки K состоит из 8 узлов замены K(1)...K(8) с памятью 64 бита каждый. Поступающий на блок подстановки 32-разрядный вектор разбивается на 8 последовательно идущих 4-

разрядных вектора, каждый из который преобразуется в 4-разрядный вектор соответствующим узлом замены, представляющим из себя таблицу из 16 целых чисел в диапазоне 0...15. Входной вектор определяет адрес строки в таблице, число из которой является выходным вектором. Затем 4-разрядные векторы последовательно объединяются в 32-разрядный выходной.

Вторая операция – циклический сдвиг влево 32-разрядного вектора, полученного в результате подстановки K. 64-разрядный блок зашифрованных данных T представляется в виде T=A(32)B(32).

Остальные блоки открытых данных в режиме простой замены зашифровываются аналогично. Следует учитывать, что данный режим шифрования обладает ограниченной криптостойкостью.

2. Гаммирование. Открытые данные, разбитые на 64-разрядные блоки T(i) (i=1,2,...,m) (m определяется объемом шифруемых данных), зашифровываются в режиме гаммирования путем поразрядного сложения по модулю 2 с гаммой шифра $Г_ш$, которая вырабатывается блоками по 64 бит, т.е.

$Г_ш$=(Г(1),Г(2),....,Г(m)).

Уравнение шифрования данных в режиме гаммирования может быть представлено в следующем виде:

Ш(i)=A(Y(i-1) ⊕ C2, Z(i-1)) {+} C(1) ⊕ T(i)=Г(i) ⊕ T(i)

В этом уравнении Ш(i) обозначает 64-разрядный блок зашифрованного текста, A - функцию шифрования в режиме простой замены (аргументами этой функции являются два 32-разрядных числа). C1 и C2 - константы, заданные в ГОСТ 28147-89. Величины y(i) и Z(i) определяются итерационно по мере формирования гаммы следующим образом:

(Y(0),Z(0))=A(S), S - 64-разрядная двоичная последовательность

(Y(i),Z(i))=(Y(i-1) [+] C2, Z(i-1) {+} C(1)), i=1, 2, ..., m.

64-разрядная последовательность, называемая синхропосылкой, не является секретным элементом шифра, но ее наличие необходимо как на передающей стороне, так и на приемной.

2. Гаммирование с обратной связью очень похож на режим гаммирования. Как и в режиме гаммирования открытые данные, разбитые на 64-разрядные

блоки Т(i), зашифровываются путем поразрядного сложения по модулю 2 с гаммой шифра $Г_ш$, которая вырабатывается блоками по 64 бит:

$Г_ш$=(Г(1), Г(2), ..., Г(m)).

Уравнение шифрования данных в режиме гаммирования с обратной связью выглядят следующим образом:

Ш(1)=А(S)⊕Т(1)=Г(1)⊕Т(1),

Ш(i)=А(Ш(i-1)⊕Т(i)=Г(i)⊕Т(i), i=2, 3, ..., m.

В ГОСТ 28147-89 определяется процесс выработки имитовставки, который единообразен для всех режимов шифрования. Имитовставка – это блок из р бит (имитовставка $И_р$), который вырабатывается либо перед шифрованием всего сообщения. либо параллельно с шифрованием по блокам. Параметр р выбирается в соответствии с необходимым уровнем имитозащищенности.

Для получения имитовставки открытые данные представляются также в виде блоков по 64 бит. Первый блок открытых данных Т(1) подвергается преобразованию, соответствующему первым 16 циклам алгоритма режима простой замены. Причем в качестве ключа используется тот же ключ, что и для шифрования данных. Полученное 64-разрядно число суммируется с открытым блоком Т(2) и сумма вновь подвергается 16 циклам шифрования для режима простой замены. Данная процедура повторятся для всех m блоков сообщения. Из полученного 64-разрядного числа выбирается отрезок Ир длиной р бит.

Имитовставка передается по каналу связи после зашифрованных данных. На приемной стороне аналогичным образом из принятого сообщения выделяется? имитовставка и сравнивается с полученной откуда?. В случае несовпадения имитовставок сообщение считается ложным.

1.4.2 Обзор ассиметричных криптосистем

Рассматриваются асимметричные алгоритмы: система с открытым ключом RSA, криптосистема Эль-Гамаля и криптосистемы на основе эллиптических уравнений.

1.4.2.1 Система с открытым ключом RSA

Наиболее популярная система с открытым ключом – криптосистема RSA, была разработана в 1977 году и получила название в честь ее создателей: Рона Ривеста (Rivest), Ади Шамира (Shamir) и Леонарда Адлемана (Adleman). Алгоритм RSA стал первым полноценным алгоритмом с открытым ключом, который может работать как в режиме шифрования данных, так и в режиме электронно-цифровой подписи [3. С. 60-69].

Они воспользовались тем фактом, что нахождение больших простых чисел в вычислительном отношении осуществляется легко, но разложение на множители произведения двух таких чисел практически невыполнимо. Доказано (теорема Рабина), что раскрытие шифра RSA эквивалентно такому разложению. Поэтому для любой длины ключа можно дать нижнюю оценку числа операций для раскрытия шифра, а с учетом производительности современных компьютеров оценить и необходимое на это время.

Возможность гарантированно оценить защищенность алгоритма RSA стала одной из причин популярности этой системы с открытым ключом на фоне десятков других схем. Поэтому алгоритм RSA используется в банковских компьютерных сетях, особенно для работы с удаленными клиентами (обслуживание кредитных карточек).

В настоящее время алгоритм RSA используется во многих стандартах, среди которых SSL, S-HHTP, S-MIME, S/WAN, STT и PCT.

Первым этапом любого асимметричного алгоритма является создание пары ключей: открытого и закрытого и распространение открытого ключа «по всему миру». Для алгоритма RSA этап создания ключей состоит из следующих операций:

1. Выбираются два простых (!) числа p и q.

2. Вычисляется их произведение n(=p*q).

3. Выбирается произвольное число e (e<n), такое, что НОД(e,(p-1)(q-1))=1, то есть e должно быть взаимно простым с числом (p-1)(q-1).

4. Методом Евклида решается в целых числах (!) уравнение e*d+(p-1)(q-1)*y=1. Здесь неизвестными являются переменные d и y – метод Евклида как раз и находит множество пар (d, y), каждая из которых является решением уравнения в целых числах.

5. Два числа (e, n) – публикуются как открытый ключ.

6. Число d хранится в строжайшем секрете – это и есть закрытый ключ, который позволит читать все послания, зашифрованные с помощью пары чисел (e, n).

Как же производится собственно шифрование с помощью этих чисел:

1. Отправитель разбивает свое сообщение на блоки, равные $k=[\log_2(n)]$ бит, где квадратные скобки обозначают взятие целой части от дробного числа.

2. Подобный блок может быть интерпретирован как число из диапазона $(0;2^k-1)$. Для каждого такого числа (назовем его m_i) вычисляется выражение $c_i=((m_i)^e) \bmod n$. Блоки c_i и есть зашифрованное сообщение Их можно спокойно передавать по открытому каналу, поскольку.операция возведения в степень по модулю простого числа, является необратимой математической задачей. Обратная ей задача носит название «логарифмирование в конечном поле» и является на несколько порядков более сложной задачей. То есть даже если злоумышленник знает числа e и n, то по c_i прочесть исходные сообщения m_i он не может никак, кроме как полным перебором m_i.

А вот на приемной стороне процесс расшифрования все же возможен, и поможет нам в этом хранимое в секрете число d. Достаточно давно была доказана теорема Эйлера, частный случай которой утверждает, что если число n представимо в виде двух простых чисел p и q, то для любого x имеет место равенство $(x^{(p-1)(q-1)}) \bmod n=1$. Для расшифрования RSA-сообщений воспользуемся этой формулой. Возведем обе ее части в степень (-y) : $(x^{(-y)(p-1)(q-1)}) \bmod n = 1^{(-y)} = 1$. Теперь умножим обе ее части на x: $(x^{(-y)(p-1)(q-1)+1}) \bmod n = 1*x = x$.

А теперь вспомним каким образом создавались открытый и закрытый ключи. С помощью алгоритма Евклида подбиралась d такое, что e*d+(p-1)(q-1)*y=1, то есть e*d=(-y)(p-1)(q-1)+1. А следовательно в последнем

выражении предыдущего абзаца можно заменить показатель степени на число (e*d). Получаем $(x^{e*d})\bmod n=x$. То есть для того чтобы прочесть сообщение $c_i=((m_i)^e)\bmod n$ достаточно возвести его в степень d по модулю m: $((c_i)^d)\bmod n = ((m_i)^{e*d})\bmod n = m_i$.

На самом деле операции возведения в степень больших чисел достаточно трудоемки для современных процессоров, даже если они производятся по оптимизированным по времени алгоритмам. Поэтому обычно весь текст сообщения кодируется обычным блочным шифром (намного более быстрым), но с использованием ключа сеанса, а вот сам ключ сеанса шифруется как раз асимметричным алгоритмом с помощью открытого ключа получателя и помещается в начало файла.

1.4.2.2 Криптосистема Эль-Гамаля

Данная система является альтернативой RSA и при равном значении ключа обеспечивает ту же криптостойкость [7. С. 56].

В отличие от RSA метод Эль-Гамаля основан на проблеме дискретного логарифма. Этим он похож на алгоритм Диффи-Хелмана. Если возводить число в степень в конечном поле достаточно легко, то восстановить аргумент по значению (то есть найти логарифм) довольно трудно.

Основу системы составляют параметры p и g – числа, первое из которых – простое, а второе – целое.

Участник А генерирует секретный ключ a и вычисляет открытый ключ $y = g^a \bmod p$. Если участник В хочет послать участнику А сообщение *m*, то он выбирает случайное число *k*, меньшее p и вычисляет

$y_1 = g^k \bmod p$ и

$y_2 = m \oplus y^k$,

где $\oplus$ означает побитовое сложение по модулю 2. Затем участник В посылает (y_1, y_2) участнику А.

Участник А, получив зашифрованное сообщение, восстанавливает его:

$m = (y_1^a \bmod p) \oplus y_2$.

Алгоритм цифровой подписи DSA, разработанный NIST (National Institute of Standard and Technology) и являющийся частью стандарта DSS частично опирается на рассмотренный метод.

1.4.2.3 Криптосистемы на основе эллиптических уравнений

Эллиптические кривые – математический объект, который может определен над любым полем (конечным, действительным, рациональным или комплексным). В криптографии обычно используются конечные поля. Эллиптическая кривая есть множество точек (x, y), удовлетворяющее следующему уравнению [7. С. 69]:

$y^2 = x^3 + ax + b$,

а также бесконечно удаленная точка. Для точек на кривой довольно легко вводится операция сложения, которая играет ту же роль, что и операция умножения в криптосистемах RSA и Эль-Гамаля.

В реальных криптосистемах на базе эллиптических уравнений используется уравнение

$y^2 = x^3 + ax + b \bmod p$, где p – простое.

Проблема дискретного логарифма на эллиптической кривой состоит в следующем: дана точка G на эллиптической кривой порядка r (количество точек на кривой) и другая точка Y на этой же кривой. Нужно найти единственную точку x такую, что Y = xG, то есть Y есть x-я степень G.

1.5 Сравнительные характеристики алгоритмов шифрования

Программная реализация криптоалгоритмов типа RSA значительно сложнее и менее производительна, чем реализация классических криптоалгоритмов типа DES. Вследствие сложности реализации операций модульной арифметики криптоалгоритм RSA часто используют только для шифрования небольших объемов информации, например для рассылки классических секретных ключей или в алгоритмах цифровой подписи, а

основную часть пересылаемой информации шифруют с помощью симметричных алгоритмов [5. С. 105].

В табл. 5 приведены некоторые сравнительные характеристики классического криптоалгоритма DES и криптоалгоритма RSA.

Табл. 5. Сравнительные характеристики DES и RSA

Характеристика	**DES**	**RSA**
Скорость шифрования	Высокая	Низкая
Используемая функция шифрования	Перестановка и подстановка	Возведение в степень
Длина ключа	56 бит	Более 500 бит
Наименее затратный криптоанализ (его сложность определяет стойкость алгоритма)	Перебор по всему ключевому пространству	Разложение числа на простые множители
Время генерации ключа	Миллисекунды	Минуты
Тип ключа	Симметричный	Асимметричный

1.6 Наиболее устойчивые к взлому алгоритмы шифрования

Следующие разработки всемирно признаны стойкими алгоритмами и публикаций о универсальных методах их взлома в средствах массовой информации не встречалось (Табл. 6) [19. С. 33].

Табл. 6. Устойчивые к взлому алгоритмы шифрования

Название алгоритма	**Автор**	**Размер блока**	**Длина ключа**
IDEA	Xuejia Lia and James Massey	64 бита	128 бит
CAST128		64 бита	128 бит
BlowFish	Bruce Schneier	64 бита	128 – 448 бит
ГОСТ	НИИ ***	64 бита	256 бит
TwoFish	Bruce Schneier	128 бит	128 – 256 бит
MARS	Корпорация IBM	128 бит	128 – 1048 бит

Криптоалгоритм именуется идеально стойким, если прочесть зашифрованный блок данных можно только перебрав все возможные ключи, до тех пор, пока сообщение не окажется осмысленным. Так как по теории вероятности искомый ключ будет найден с вероятностью 1/2 после перебора половины всех ключей, то на взлом идеально стойкого криптоалгоритма с ключом длины N потребуется в среднем 2^{N-1} проверок. Таким образом, в общем случае стойкость блочного шифра зависит только от длины ключа и возрастает экспоненциально с ее ростом. Даже предположив, что перебор ключей производится на специально созданной многопроцессорной системе, в которой благодаря диагональному параллелизму на проверку 1 ключа уходит только 1 такт, то на взлом 128 битного ключа современной технике потребуется не менее 10^{21} лет. Естественно, все сказанное относится только к идеально стойким шифрам, которыми, например, с большой долей уверенности являются приведенные в таблице выше алгоритмы.

Кроме этого условия к идеально стойким криптоалгоритмам применяется еще одно очень важное требование, которому они должны обязательно соответствовать. При известных исходном и зашифрованном значениях блока ключ, которым произведено это преобразование, можно узнать также только

полным перебором. Ситуации, в которых постороннему наблюдателю известна часть исходного текста встречаются повсеместно. Это могут быть стандартные надписи в электронных бланках, фиксированные заголовки форматов файлов, довольно часто встречающиеся в тексте длинные слова или последовательности байт.

1.7 Идентификация и аутентификация пользователей

Расширение круга пользователей распределенных систем и усложнение задач, решаемых с помощью подобных сетей, привели к осознанию первостепенной важности проблем, связанных с обеспечением информационной безопасности при межсетевом взаимодействии, прежде всего с использованием открытых каналов передачи данных, создающих потенциальную возможность для действий как пассивного нарушителя, имеющего возможность только просматривать доступные ему сообщения, так и активного нарушителя, который наряду с прослушиванием может модифицировать доступные ему сообщения [6. С. 120-135].

Вот почему одной из важнейших задач обеспечения безопасности является разработка методов и средств, позволяющих одной стороне (проверяющему) убедиться в идентичности другой стороны (доказывающего).

Прежде чем получить доступ к ресурсам компьютерной системы, пользователь должен пройти процесс представления компьютерной системе, который включает две стадии:

- идентификацию – пользователь сообщает системе по ее запросу свое имя (идентификатор);
- аутентификацию – пользователь подтверждает идентификацию, вводя в систему уникальную, не известную другим пользователям информацию о себе (например, пароль).

С точки зрения проверяющего к аутентификации предъявляются следующие критичные требования:

- при взаимном доверии сторон A и B стороне A необходимо убедиться в идентичности стороны B;
- сторона B не должна иметь возможность повторного обмена информацией, переданной ранее в ходе взаимодействия со стороной A, с целью выдать себя за сторону A при аутентификации со стороны C;
- различия в аутентификационном обмене между сторонами должны быть настолько существенными, чтобы ни одна из сторон не смогла реализовать обмен данными от имени другой стороны;
- информация пересылаемая в рамках аутентификационного обмена, должна препятствовать получению статистических сведений, на основе которых пассивный нарушитель смог бы имитировать обмен от имени какой-либо стороны, т.е. в результате информационного обмена знания участников друг о друге не должны увеличиваться.

Все аутентификации могут быть разделены на следующие категории [3. С. 91-98]:

- на основе знания чего-либо. Примером могут служить стандартные пароли, персональные идентификационные номера (PIN), а также секретные и открытые ключи;
- на основе обладания чем-либо. Обычно это магнитные карты, смарт-карты, touch memory и персональные генераторы, которые используются для создания одноразовых паролей.
- на основе каких-либо неотъемлемых характеристик. Эта категория включает методы, базирующиеся на проверке пользовательских биометрических характеристик (голос, сетчатка глаза, отпечатки пальцев). В данной категории не используются криптографические методы и средства. Аутентификация на основе биометрических характеристик применяется для контроля доступа в помещения либо к какой-либо технике.

В табл. 7 приведены основные типы аутентификации пользователей

Табл. 7. Основные типы аутентификации пользователей

Тип	Описание
Пароли или PIN-коды (персональные идентификационные номера)	Что-то, что знает пользователь и что также знает другой участник взаимодействия. Обычно аутентификация производится в 2 этапа. Может организовываться обмен паролями для взаимной аутентификации.
Одноразовый пароль	Пароль, который никогда больше не используется. Часто используется постоянно меняющееся значение, которое базируется на постоянном пароле.
CHAP (протокол аутентификации запрос-ответ)	Одна из сторон инициирует аутентификацию с помощью посылки уникального и непредсказуемого значения «запрос» другой стороне, а другая сторона посылает вычисленный с помощью "запроса" и секрета ответ. Так как обе стороны владеют секретом, то первая сторона может проверить правильность ответа второй стороны.
Встречная проверка (Callback)	Телефонный звонок серверу и указание имени пользователя приводит к тому, что сервер затем сам звонит по номеру, который указан для этого имени пользователя в его конфигурационных данных.

Говоря о классификации по уровню обеспечиваемой безопасности, необходимо упомянуть о типовых атаках на аутентификацию и общих методах, помогающих их избежать (табл. 8) [19. С. 152-159]

Табл. 8. Типовые атаки на аутентификацию и их пути обхождения

Тип атаки	Пути обхождения
1. Повторная передача. Заключается в повторной передаче аутентификационных данных каким-либо пользователем	Использование протоколов типа «запро-ответ»; использование меток времени или случайных чисел; вставка идентифицирующей информации в ответы
2. Подмена стороны аутентификационного обмена. Злоумышленник в ходе данной атаки участвует в процессе аутентификационного обмена между двумя сторонами и имеет возможность модификации проходящего через него трафик	Связывание всех сообщений в рамках данной сессии (например, используя одноразовый и уникальный идентификатор во всех сообщениях)
3. Отражение передачи. Один из вариантов предыдущей атаки, в ходе которой злоумышленник в рамках данной сессии пересылает обратно перехваченную информацию	Использование идентификаторов сторон в передаваемых сообщения; построение формы таки образом, чтобы каждое сообщение имело отличную от других форму
4. Выборка текста. Злоумышленник перехватывает аутентификационный трафик и пытается получить информацию о долговременных ключах	Использование протоколов, обладающих свойством доказательства нулевым знанием; включение в сообщение случайных чисел
5. Задержка передачи. Злоумышленник перехватывает некоторую информацию и передает ее спустя некоторое время	Комбинирование использования меток времени вместе со случайными числами

Самая опасная угроза аутентификации возникает в том случае, когда нарушитель выдает себя за какую-либо другую сторону, обычно обладающими существенными привилегиями в системе. Именно получение привилегий и возможность работы от лица другого пользователя (в случае, если в системе используется система аудита) являются основными целями злоумышленника.

Учитывая описанные выше атаки, а также вопросы практического применения, можно выделить следующие свойства аутентификации [4. С. 134-137]:

- взаимная аутентификация. Свойство, отражающее необходимость обоюдной аутентификации между сторонами аутентификационного обмена;
- вычислительная эффективность. Количество операций, необходимых для выполнения аутентификации;
- коммуникационная эффективность. Данное свойство отражает количество сообщений и их длину, необходимую для осуществления аутентификации;
- наличие третьей стороны. Примеров третьей стороны может служить доверенный сервер распределения симметричных ключей или сервер, реализующий дерево сертификатов для распределения открытых ключей;
- основа гарантий безопасности. Примером могут служить протоколы, обладающие свойством доказательства с нулевым знанием;
- хранение секрета. Здесь имеется в виду, каким способом реализовано хранение критичной ключевой информации.

Одной из основных задач, решаемых во время работы подобных протоколов, является разграничение доступа к ресурсам, когда при привилегированном доступе необходимо убедиться в идентичности пользователей (например, локальный или удаленный доступ к ресурсам компьютера или доступ к приложениям со стороны пользователей).

1.8 Электронно-цифровая подпись

Последние несколько лет ознаменовались постепенной заменой бумажной технологии обработки информации ее электронным аналогом. Со временем можно ожидать полного вытеснения бумажного документооборота электронным [26. С. 156-160]. Однако представление традиционных бумажных документов в виде электронных последовательностей, состоящих из нулей и единиц, обезличивает последние. Защитных атрибутов бумажных документов: подписей, печатей и штампов, водяных знаков, специальной фактуры бумажной поверхности, - у электронного представления документов нет. Но электронные документы нужно защищать не менее тщательно, чем бумажные. Поэтому возникает задача разработки такого механизма электронной защиты, который бы смог заменить подпись и печать на бумажных документах. То есть необходимо разработать механизм электронно-цифровой подписи, которая представляет собой дополнительную информацию, приписываемую к защищаемым данным. Цифровая подпись зависит от содержания подписываемого документа и некоего секретного элемента (ключа), которым обладает только лицо, участвующее в защищенном обмене.

Такой механизм обеспечивает:

- Во-первых, цифровая подпись должна подтверждать, что подписывающее лицо не случайно подписало электронный документ.
- Во-вторых, цифровая подпись должна подтверждать, что только подписывающее лицо, и только оно, подписало электронный документ.
- В-третьих, цифровая подпись должна зависеть от содержания подписываемого документа и времени его подписания.
- В-четвертых, подписывающее лицо не должно иметь возможности в последствии отказаться от факта подписи документа.

1.8.1 Электронно-цифровая подпись при помощи симметричных криптосистем

Первые варианты цифровой подписи были реализованы при помощи симметричных криптосистем, в которой абоненты, участвующие в обмене

сообщениями, используют один и тоже секретный ключ для простановки и проверки подписи под документом. В качестве алгоритма криптографического преобразования может использоваться любая из симметричных криптосистем, обладающая специальными режимами функционирования (например, DES, ГОСТ 28147-89 и т.п.) [3. С. 123-130].

Процедура подписи документов при помощи симметричной криптосистемы может выглядеть следующим образом:

1. Отправитель А и получатель В обладают одинаковым секретным ключом К.

2. Отправитель А зашифровывает цифровое сообщение, используя ключ К, и посылает зашифрованное сообщение получателю В. При этом, как уже отмечалось, используется криптосистема, обладающая специальными механизмами функционирования, например ГОСТ 28147-89 в режиме выработки имитовставки.

3. Получатель В расшифровывает сообщение при помощи ключа К.

Так как только пользователи А и В обладают секретным ключом, то тем самым обеспечивается гарантия, что сообщение подписано именно пользователем А, а не злоумышленником С. Однако, данная схема применима только в тех сетях, в которых можно дать стопроцентную гарантирую надежности каждого из абонентов, т.к. в обратном случае существует потенциальная возможность мошенничества со стороны одного из абонентов, владеющих секретным ключом.

Для устранения указанного недостатка была предложена схема с доверенным арбитром. В данной схеме помимо пользователей А и В существует арбитр – Х. Х может обмениваться сообщениями и с отправителем А и с получателем В, и владеет секретными ключами обоих - КА и КВ. Процедура подписания документов в данной схеме выглядит следующим образом:

1. Отправитель А зашифровывает сообщение, используя ключ КА, и посылает его арбитру Х.

2. Арбитр Х расшифровывает сообщение, также используя ключ КА.

3. Расшифрованное сообщение арбитр X зашифровывает на ключе получателя B KB.

4. Данное сообщение арбитр X посылает получателю B.

5. Получатель B расшифровывает сообщение, полученное от арбитра X, используя ключ KB.

Насколько эффективна данная схема? Рассмотрим ее с учетом вышеуказанных требований, предъявляемых к цифровой подписи:

- Во-первых, арбитр X и получатель B знают, что сообщение пришло именно от отправителя A. Уверенность арбитра X основана на том, что секретный ключ KA имеют только отправитель A и арбитр X. Подтверждение арбитра X служит доказательством получателю B.
- Во-вторых, только отправитель A знает ключ KA (и арбитр X, как доверенный арбитр). Арбитр X может получить зашифрованное сообщение на ключе KA только от отправителя A. Если злоумышленник C пытается послать сообщение от имени отправитель A, то арбитр X сразу обнаружит это на 2-м шаге.
- В-третьих, если получатель B пытается использовать цифровую подпись, полученную от арбитра X, и присоединить ее к другому сообщению, выдавая его за сообщение от отправителя A, это выявляется. Арбитр X может потребовать от получателя B данное сообщение и затем сравнивает его с сообщением, зашифрованном на ключе KA. Сразу выявляется факт несоответствия между двумя сообщениями. Если бы получатель B попытался модифицировать присланное сообщение, то арбитра X аналогичным способом выявила бы подделку.
- В-четвертых, даже если отправитель A отрицает факт посылки подписанного сообщения, подтверждение от арбитра X говорит об обратном.

Можно заметить, что самым критичным звеном этой схемы является именно арбитр X. Во-первых, он должен быть действительно независимой ни от кого стороной. А во-вторых, арбитр X должен быть абсолютно безошибочным. Ошибка в рассмотрении даже одной из нескольких тысяч спорных ситуаций подорвет доверие не только к арбитру X, но и ко всем

предыдущим подписанным документам, достоверность которых удостоверялась арбитром X.

Именно поэтому, несмотря на теоретическую возможность применения симметричных криптосистем, на практике для выработки подписи они не используются, поскольку требуют дорогостоящих и громоздких мероприятий по поддержанию достаточного уровня достоверности передаваемых данных.

1.8.2 Электронно-цифровая подпись при помощи асимметричных криптосистем

В таких криптосистемах нет необходимости и в арбитрах. Суть этого подхода заключается в том, что в обмене защищаемыми документами каждый абонент использует пару взаимосвязанных ключей – открытый и секретный. Отправитель подписываемого документа передает получателю открытый ключ. Он может это сделать любым несекретным способом или поместить ключ в общедоступный справочник. При помощи открытого ключа получатель проверяет подлинность получаемой информации. Секретный ключ, при помощи которого подписывалась информация, хранится в тайне от всех [5. С. 142-148].

Можно заметить, что в данной схеме абоненты используют различные ключи, что не позволяет мошенничать ни одной из сторон. Подробный анализ цифровой подписи на основе криптосистем с открытыми ключами показывает, что она полностью удовлетворяет требованиям, предъявляемым к ней. В настоящий момент широкого известны цифровые подписи, построенные по алгоритмам RSA, Эль-Гамаля, Шнорра, Рабина и математического аппарата эллиптических кривых.

В табл. 9 представлены основные типы электронно-цифровой подписи. Добавление временных меток позволяет обеспечить ограниченную форму контроля участников взаимодействия.

Табл. 9. Основные типы электронно-цифровой подписи

Тип	Комментарии
DSA (Digital Signature Authorization)	Алгоритм с использованием открытого ключа для создания электронной подписи, но не для шифрования. Секретное создание хэш-значения и публичная проверка ее - только один человек может создать хэш-значение сообщения, но любой может проверить ее корректность. Основан на вычислительной сложности взятия логарифмов в конечных полях.
RSA	Запатентованная RSA электронная подпись, которая позволяет проверить целостность сообщения и личность лица, создавшего электронную подпись. Отправитель создает хэш-функцию сообщения, а затем шифрует ее с использованием своего секретного ключа. Получатель использует открытый ключ отправителя для расшифровки хэша, сам рассчитывает хэш для сообщения, и сравнивает эти два хэша.
MAC (код аутентификации сообщения)	Электронная подпись, использующая схемы хэширования, аналогичные MD или SHA, но хэш-значение вычисляется с использованием как данных сообщения, так и секретного ключа.
DTS (служба электронных временных меток)	Выдает пользователям временные метки, связанные с данными документа, криптографически стойким образом.

1.9 Хеширование паролей

Хеширования паролей – это метод, позволяющей пользователям запоминать не 128 байт, то есть 256 шестнадцатеричных цифр ключа, а некоторое осмысленное выражение, слово или последовательность символов, называющуюся паролем. Действительно, при разработке любого криптоалгоритма следует учитывать, что в половине случаев конечным пользователем системы является человек, а не автоматическая система. Это ставит вопрос о том, удобно, и вообще реально ли человеку запомнить 128-битный ключ (32 шестнадцатеричные цифры). На самом деле предел запоминаемости лежит на границе 8-12 подобных символов, а, следовательно, если мы будем заставлять пользователя оперировать именно ключом, тем самым мы практически вынудим его к записи ключа на каком-либо листке бумаги или электронном носителе, например, в текстовом файле. Это, естественно, резко снижает защищенность системы [3. С. 72-75].

Для решения этой проблемы были разработаны методы, преобразующие произносимую, осмысленную строку произвольной длины – пароль, в указанный ключ заранее заданной длины. В подавляющем большинстве случаев для этой операции используются так называемые хеш-функции (от англ. hashing – мелкая нарезка и перемешивание). Хеш-функцией называется такое математическое или алгоритмическое преобразование заданного блока данных, которое обладает следующими свойствами (иногда хеш-функцию называют дайджестом сообщений – MD):

- хеш-функция имеет бесконечную область определения,
- хеш-функция имеет конечную область значений,
- она необратима,
- изменение входного потока информации на один бит меняет около половины всех бит выходного потока, то есть результата хеш-функции.

Эти свойства позволяют подавать на вход хеш-функции пароли, то есть текстовые строки произвольной длины на любом национальном языке и, ограничив область значений функции диапазоном $0..2^N-1$, где N – длина ключа в битах, получать на выходе достаточно равномерно распределенные по области значения блоки информации – ключи.

Нетрудно заметить, что требования, подобные 3 и 4 пунктам требований к хеш-функции, выполняют блочные шифры. Это указывает на один из возможных путей реализации стойких хеш-функций – проведение блочных криптопреобразований над материалом строки-пароля. Этот метод и используется в различных вариациях практически во всех современных криптосистемах. Материал строки-пароля многократно последовательно используется в качестве ключа для шифрования некоторого заранее известного блока данных – на выходе получается зашифрованный блок информации, однозначно зависящий только от пароля и при этом имеющий достаточно хорошие статистические характеристики. Такой блок или несколько таких блоков и используются в качестве ключа для дальнейших криптопреобразований [7. С. 82].

Характер применения блочного шифра для хеширования определяется отношением размера блока используемого криптоалгоритма и разрядности требуемого хеш-результата.

Если указанные выше величины совпадают, то используется схема одноцепочечного блочного шифрования. Первоначальное значение хеш-результата H_0 устанавливается равным 0, вся строка-пароль разбивается на блоки байт, равные по длине ключу используемого для хеширования блочного шифра, затем производятся преобразования по реккурентной формуле:

$H_j=H_{j-1}$ XOR EnCrypt(H_{j-1},PSW_j),

где EnCrypt(X,Key) – используемый блочный шифр (рис.14).

Последнее значение H_k используется в качестве искомого результата.

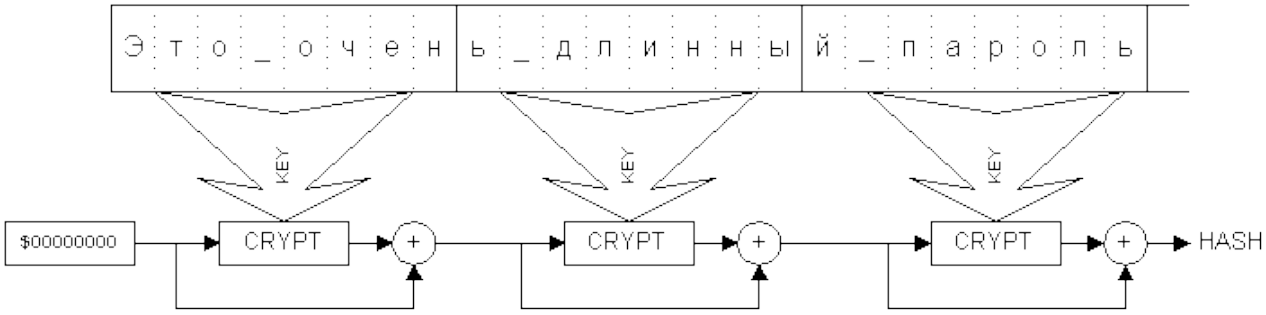


Рис. 14. Метод преобразования хеш-функции

В том случае, когда длина ключа ровно в два раза превосходит длину блока, а подобная зависимость довольно часто встречается в блочных шифрах, используется схема, напоминающая сеть Фейштеля. Характерным недостатком и приведенной выше формулы, и хеш-функции, основанной на сети Фейштеля, является большая ресурсоемкость в отношении пароля. Для проведения только одного преобразования, например, блочным шифром с ключом длиной 128 бит используется 16 байт строки-пароля, а сама длина пароля редко превышает 32 символа. Следовательно, при вычислении хеш-функции над паролем будут произведено максимум 2 «полноценных» криптопреобразования [11. С. 212-215].

Решение этой проблемы можно достичь двумя путями: 1) предварительно «размножить» строку-пароль, например, записав ее многократно последовательно до достижения длины, скажем, в 256 символов; 2) модифицировать схему использования криптоалгоритма так, чтобы материал строки-пароля «медленнее» тратился при вычислении ключа.

По второму пути пошли исследователи Девис и Майер, предложившие алгоритм также на основе блочного шифра, но использующий материал строки-пароля многократно и небольшими порциями [26. С. 25-35]. В нем просматриваются элементы обеих приведенных выше схем, но криптостойкость этого алгоритма подтверждена многочисленными реализациями в различных криптосистемах. Алгоритм получил название «Tandem DM» (рис.15):

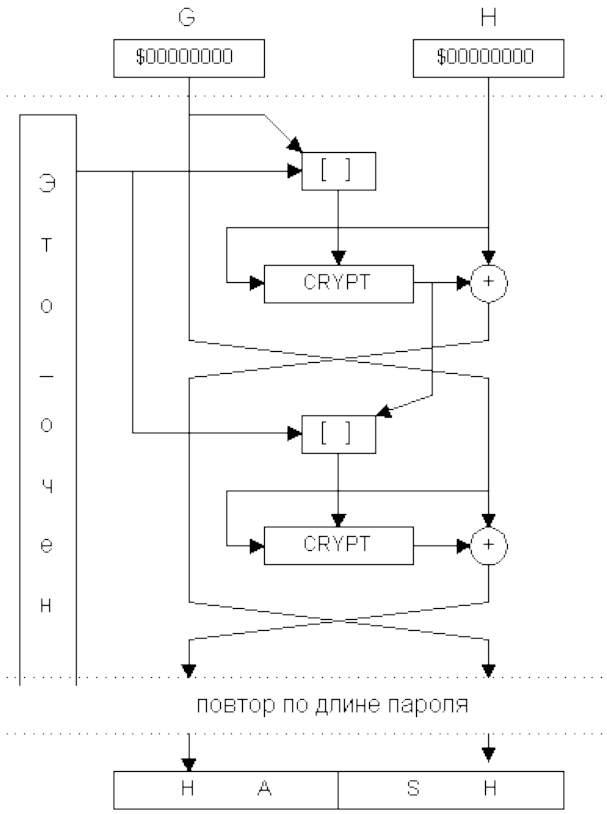


Рис. 15. Алгоритм Tandem DM

$G_0=0$; $H_0=0$;

FOR J = 1 TO N DO

BEGIN

TMP=EnCrypt(H,[G,PSW_j]); H'=H XOR TMP;

TMP=EnCrypt(G,[PSW_j,TMP]); G'=G XOR TMP;

END;

KEY= [G_k,H_k]

Квадратными скобками (X16=[A8,B8]) здесь обозначено простое объединение (склеивание) двух блоков информации равной величины в один – удвоенной разрядности. А в качестве процедуры EnCrypt(X,Key) опять может быть выбран любой стойкий блочный шифр. Как видно из формул, данный алгоритм ориентирован на то, что длина ключа двукратно превышает размер блока криптоалгоритма. А характерной особенностью схемы является тот факт, что строка пароля считывается блоками по половине длины ключа, и каждый блок используется в создании хеш-результата дважды. Таким образом, при длине пароля в 20 символов и необходимости создания 128 битного ключа внутренний цикл хеш-функции повторится 3 раза.

Хэш-функции прежде всего используются для обнаружения модификации сообщения (то есть для электронной подписи). В табл. 10 приведены типы хеш-функций, распространенные в настоящее время [4. С. 26].

Табл. 10. Распространенные хеш-функции

Тип	**Описание**
MD2	Самая медленная, оптимизирована для 8-битовых машин
MD4	Самая быстрая, оптимизирована для 32-битных машин Не так давно взломана
MD5	Наиболее распространенная из семейства MD-функций. Похожа на MD4, но средства повышения безопасности делают ее на 33% медленнее, чем MD4 Обеспечивает целостность данных Считается безопасной
SHA (Secure Hash Algorithm)	Создает 160-битное значение хэш-функции из исходных данных переменного размера. Предложена NIST и принята правительством США как стандарт Предназначена для использования в стандарте DSS

1.10 Управление криптографическими ключами

Помимо выбора подходящей для конкретной информационной системы средств криптографической защиты информации, важной проблемой является управление ключами. Как бы ни была сложна и надежна сама криптосистема, она основана на использовании ключей. Если для обеспечения конфиденциального обмена информацией между двумя пользователями процесс обмена ключами тривиален, то в информационной системе, где количество пользователей составляет сотни и тысячи, управление ключами – серьезная проблема.

1.10.1 Симметричная методология

В этой методологии и для шифрования, и для расшифрования отправителем и получателем применяется один и тот же ключ, об использовании которого они договорились до начала взаимодействия. Если ключ не был скомпрометирован, то при расшифровании автоматически выполняется аутентификация отправителя, так как только отправитель имеет ключ, с помощью которого можно зашифровать информацию, и только получатель имеет ключ, с помощью которого можно расшифровать информацию. Так как отправитель и получатель - единственные люди, которые знают этот симметричный ключ, при компрометации ключа будет скомпрометировано только взаимодействие этих двух пользователей. Проблемой, которая будет актуальна и для других криптосистем, является вопрос о том, как безопасно распространять симметричные (секретные) ключи [11. С. 96-105].

Алгоритмы симметричного шифрования используют ключи не очень большой длины и могут быстро шифровать большие объемы данных.

Порядок использования систем с симметричными ключами:

1. Безопасно создается, распространяется и сохраняется симметричный секретный ключ.

2. Отправитель создает электронную подпись с помощью расчета хэш-функции для текста и присоединения полученной строки к тексту.

3. Отправитель использует быстрый симметричный алгоритм шифрования/ расшифрования вместе с секретным симметричным ключом к полученному пакету (тексту вместе с присоединенной электронной подписью) для получения зашифрованного текста. Неявно таким образом производится аутентификация, так как только отправитель знает симметричный секретный ключ и может зашифровать этот пакет. Только получатель знает симметричный секретный ключ и может расшифровать этот пакет.

4. Отправитель передает зашифрованный текст. Симметричный секретный ключ никогда не передается по незащищенным каналам связи.

5. Получатель использует тот же самый симметричный алгоритм шифрования/расшифрования вместе с тем же самым симметричным ключом (который уже есть у получателя) к зашифрованному тексту для восстановления исходного текста и электронной подписи. Его успешное восстановление аутентифицирует кого-то, кто знает секретный ключ.

6. Получатель отделяет электронную подпись от текста.

7. Получатель создает другую электронную подпись с помощью расчета хэш-функции для полученного текста.

8. Получатель сравнивает две этих электронных подписи для проверки целостности сообщения (отсутствия его искажения)

Доступными сегодня средствами, в которых используется симметричная методология, являются:

- Kerberos, который был разработан для аутентификации доступа к ресурсам в сети, а не для верификации данных. Он использует центральную базу данных, в которой хранятся копии секретных ключей всех пользователей.
- Сети банкоматов (ATM Banking Networks). Эти системы являются оригинальными разработками владеющих ими банков и не продаются. В них также используются симметричные методологии.

1.10.2 Асимметричная методология

В этой методологии ключи для шифрования и расшифрования разные, хотя и создаются вместе. Один ключ делается известным всем, а другой держится в тайне. Хотя можно шифровать и расшифровывать обоими ключами, данные, зашифрованные одним ключом, могут быть расшифрованы только другим ключом [6. С. 34-39].

Все асимметричные криптосистемы являются объектом атак путем прямого перебора ключей, и поэтому в них должны использоваться гораздо более длинные ключи, чем те, которые используются в симметричных криптосистемах, для обеспечения эквивалентного уровня защиты. Это сразу же сказывается на вычислительных ресурсах, требуемых для шифрования, хотя

алгоритмы шифрования на эллиптических кривых могут смягчить эту проблему.

В табл. 11 приведены данные об эквивалентных длинах симметричного и открытого ключей

Табл. 11. Длина симметричного и открытого ключей

Длина симметричного ключа	**Длина открытого ключа**
56 бит	384 бит
64 бита	512 бит
80 бит	768 бит
112 бит	1792 бита
128 бит	2304 бита

Для того чтобы избежать низкой скорости алгоритмов асимметричного шифрования, генерируется временный симметричный ключ для каждого сообщения и только он шифруется асимметричными алгоритмами. Само сообщение шифруется с использованием этого временного сеансового ключа и алгоритма шифрования/расшифрования. Затем этот сеансовый ключ шифруется с помощью открытого асимметричного ключа получателя и асимметричного алгоритма шифрования. После этого этот зашифрованный сеансовый ключ вместе с зашифрованным сообщением передается получателю. Получатель использует тот же самый асимметричный алгоритм шифрования и свой секретный ключ для расшифрования сеансового ключа, а полученный сеансовый ключ используется для расшифровки самого сообщения.

В асимметричных криптосистемах важно, чтобы сеансовые и асимметричные ключи были сопоставимы в отношении уровня безопасности, который они обеспечивают. Если используется короткий сеансовый ключ (например, 40-битовый DES), то не имеет значения, насколько велики асимметричные ключи. Хакеры будут атаковать не их, а сеансовые ключи. Асимметричные открытые ключи уязвимы к атакам прямым перебором отчасти из-за того, что их тяжело заменить. Если атакующий узнает секретный

асимметричный ключ, то будет скомпрометирован не только текущее, но и все последующие взаимодействия между отправителем и получателем [5. С. 45-47].

Порядок использования систем с асимметричными ключами:

1. Безопасно создаются и распространяются асимметричные открытые и секретные ключи. Секретный асимметричный ключ передается его владельцу. Открытый асимметричный ключ хранится в базе данных X.500 и администрируется центром выдачи сертификатов (по-английски - Certification Authority или CA). Подразумевается, что пользователи должны верить, что в такой системе производится безопасное создание, распределение и администрирование ключами. Более того, если создатель ключей и лицо или система, администрирующие их, не одно и то же, то конечный пользователь должен верить, что создатель ключей на самом деле уничтожил их копию.

2. Создается электронная подпись текста с помощью вычисления его хэш-функции. Полученное значение шифруется с использованием асимметричного секретного ключа отправителя, а затем полученная строка символов добавляется к передаваемому тексту (только отправитель может создать электронную подпись).

3. Создается секретный симметричный ключ, который будет использоваться для шифрования только этого сообщения или сеанса взаимодействия (сеансовый ключ), затем при помощи симметричного алгоритма шифрования/расшифрования и этого ключа шифруется исходный текст вместе с добавленной к нему электронной подписью – получается зашифрованный текст (шифр-текст).

4. Теперь нужно решить проблему с передачей сеансового ключа получателю сообщения.

5. Отправитель должен иметь асимметричный открытый ключ центра выдачи сертификатов (CA). Перехват незашифрованных запросов на получение этого открытого ключа является распространенной формой атаки. Может существовать целая система сертификатов, подтверждающих подлинность открытого ключа CA. Стандарт X.509 описывает ряд методов для получения пользователями открытых ключей CA, но ни один из них не может полностью

защитить от подмены открытого ключа CA, что наглядно доказывает, что нет такой системы, в которой можно было бы гарантировать подлинность открытого ключа CA.

6. Отправитель запрашивает у CA асимметричный открытый ключ получателя сообщения. Этот процесс уязвим к атаке, в ходе которой атакующий вмешивается во взаимодействие между отправителем и получателем и может модифицировать трафик, передаваемый между ними. Поэтому открытый асимметричный ключ получателя «подписывается» CA. Это означает, что CA использовал свой асимметричный секретный ключ для шифрования асимметричного открытого ключа получателя. Только CA знает асимметричный секретный ключ CA, поэтому есть гарантии того, что открытый асимметричный ключ получателя получен именно от CA.

7. После получения асимметричный открытый ключ получателя расшифровывается с помощью асимметричного открытого ключа CA и алгоритма асимметричного шифрования/расшифрования. Естественно, предполагается, что CA не был скомпрометирован. Если же он оказывается скомпрометированным, то это выводит из строя всю сеть его пользователей. Поэтому можно и самому зашифровать открытые ключи других пользователей, но где уверенность в том, что они не скомпрометированы?

8. Теперь шифруется сеансовый ключ с использованием асимметричного алгоритма шифрования/расшифрования и асимметричного ключа получателя (полученного от CA и расшифрованного).

9. Зашифрованный сеансовый ключ присоединяется к зашифрованному тексту (который включает в себя также добавленную ранее электронную подпись).

10. Весь полученный пакет данных (зашифрованный текст, в который входит помимо исходного текста его электронная подпись, и зашифрованный сеансовый ключ) передается получателю. Так как зашифрованный сеансовый ключ передается по незащищенной сети, он является очевидным объектом различных атак.

11. Получатель выделяет зашифрованный сеансовый ключ из полученного пакета.

12. Теперь получателю нужно решить проблему с расшифровкой сеансового ключа.

13. Получатель должен иметь асимметричный открытый ключ центра выдачи сертификатов (CA).

14. Используя свой секретный асимметричный ключ и тот же самый асимметричный алгоритм шифрования получатель расшифровывает сеансовый ключ.

15. Получатель применяет тот же самый симметричный алгоритм шифрования/расшифрования и расшифрованный симметричный (сеансовый) ключ к зашифрованному тексту и получает исходный текст вместе с электронной подписью.

16. Получатель отделяет электронную подпись от исходного текста.

17. Получатель запрашивает у CA асимметричный открытый ключ отправителя.

18. Как только этот ключ получен, получатель расшифровывает его с помощью открытого ключа CA и соответствующего асимметричного алгоритма шифрования/расшифрования.

19. Затем расшифровывается хэш-функция текста с использованием открытого ключа отправителя и асимметричного алгоритма шифрования/расшифрования.

20. Повторно вычисляется хэш-функция полученного исходного текста.

21. Две эти хэш-функции сравниваются для проверки того, что текст не был изменен.

Итак, все криптосистемы делятся на симметричные и асимметричные. Симметричные криптосистемы используют секретный ключ, который используется как для шифрования текста, так и для его расшифрования. Чем отличаются асимметричные криптосистемы, которые используют 2 ключа – открытый и закрытый. Открытый ключ нужен для шифрования текста, а закрытый – для его расшифрования. В настоящее время к наиболее популярным

симметричным криптосистемам относят блочные шифры, которые выделяются достаточной криптостойкостью и простотой процедур шифрования и расшифрования, а также генерацией ключа. Не смотря на свою простоту блочные шифры имеют высокую скорость шифрования/расшифрования файла, а также высокую гарантированную стойкость.

Блочные шифры базируются на сетях Фейштеля, которые разбивают шифруемый текст на блоки, перемешают их, используя «исключающее ИЛИ» и циклические сдвиги, а также количество раундов шифрования. Чем больше раундов применимо к блоку, тем более надежен и криптостоик он.

Одним из самых распространенных алгоритмов блочного шифрования является DES (Data Encryption Standard), который получил достаточно широкое распространение в свое время. Однако, со временем, на его место пришел наиболее усовершенствованный AES (Advanced Encryption Standard), который стал де-факто мировым криптостандартом на ближайшие 10-20 лет. Наиболее яркие его представители это MARS, Serpent, TwoFish, Rijndael и RC6, а также российский представитель стандарт шифрования данных – ГОСТ 28147-89.

Асимметричные криптосистемы не столь обширны как симметричные, т.к. они сложны и менее производительны. Яркие его представители – это система RSA, криптосистема Эль-Гамаля и криптосистема на основе эллиптических уравнений и другие.

Таким образом, симметричные криптосистемы обладают более быстрой скоростью шифрования данных, чем асимметричные, которые часто используют только для шифрования небольших объемов информации, например для рассылки классических секретных ключей или в алгоритмах цифровой подписи, а основную часть пересылаемой информации шифруют с помощью симметричных алгоритмов.

Наиболее устойчивыми к взлому алгоритмами шифрования являются IDEA, CAST128, BlowFish, TwoFish, MARS и ГОСТ.

Одна из важнейших задач обеспечения безопасности является идентификация и аутентификация пользователей, которые заключаются, например, в вводе логина и пароля. Аутентификация требуется в персональных

идентификационных номерах (PIN), магнитных карты, смарт-картах. Самая опасная угроза аутентификации возникает в том случае, когда нарушитель выдает себя за какую-либо другую сторону, обычно обладающими существенными привилегиями в системе, поэтому одной из основных задач является разграничение доступа к ресурсам, когда при привилегированном доступе необходимо убедиться в идентичности пользователей.

Для защиты электронной почты от несанкционированного прочтения и ее изменения применяются электронно-цифровые подписи, которые подтверждают своего отправителя. В электронно-цифровой подписи при помощи симметричных криптосистем участвуют отправители A, B, а также независимый доверенный арбитр X, который используя ключи обоих отправителей, расшифровывает и зашифровывает сообщение и отсылает его назад отправителям. Однако на практике симметричные криптосистемы для выработки подписи не используются, поскольку требуют дорогостоящих и громоздких мероприятий по поддержанию достаточного уровня достоверности передаваемых данных. Поэтому в настоящее время используются менее замысловатые асимметричные криптосистемы. В них абоненты используют пару взаимосвязанных ключей – открытый и секретный. Отправитель зашифровывает сообщение открытым ключом, а получатель расшифровывает его используя только известный ему – секретный ключ.

Метод хеширования паролей позволяет преобразовать длинный пароль в ключ заранее заданной длины. В подавляющем большинстве случаев для этой операции используются так называемые хеш-функции. На входе хеш-функции задаются пароли, то есть текстовые строки произвольной длины в битах, а на выходе достаточно равномерно распределенные по области значения блоки информации – ключи.

Управление криптографическими ключами является серьезной проблемой когда количество пользователей составляет сотни и тысячи. Как было сказано выше, алгоритмы симметричного шифрования используют одинаковые ключи для шифрования и расшифрования данных. Доступными сегодня средствами, в которых используется симметричная методология, являются Kerberos, который

был разработан для аутентификации доступа к ресурсам в сети и сети банкоматов (ATM Banking Networks), в которых используются симметричные методологии.

В асимметричном шифрования для того чтобы избежать низкой скорости алгоритмов, генерируется временный симметричный ключ для каждого сообщения и только он шифруется асимметричными алгоритмами. Само сообщение шифруется с использованием этого временного сеансового ключа и алгоритма шифрования/расшифрования. Затем этот сеансовый ключ шифруется с помощью открытого асимметричного ключа получателя и асимметричного алгоритма шифрования. После этого этот зашифрованный сеансовый ключ вместе с зашифрованным сообщением передается получателю. Получатель использует тот же самый асимметричный алгоритм шифрования и свой секретный ключ для расшифровки сеансового ключа, а полученный сеансовый ключ используется для расшифровки самого сообщения.

2. ПРОЕКТИРОВАНИЕ МОДИФИЦИРОВАННОГО АЛГОРИТМА RC6

В данной работе описывается разработка алгоритма шифрования, основанная на стандартном алгоритме RC6, как одном из широко распространенных в настоящее время.

Данный алгоритм шифрования обладает следующими свойствами:

1. Алгоритм RC6 является алгоритмом AES по криптостойкости шифра и успешно противостоит всем широко известным методам криптоанализа (при условии длины ключа 512 и 1024 бита).

2. Шифрование и расшифрование текста происходит за сравнительно быстрый период времени, т.к. используются сети Фейштеля и ассемблерные вставки.

3. Зашифрованный текст поддается чтению только при наличии ключа.

4. Знание алгоритма шифрования не снижает криптостойкость шифра.

5. Число операций, необходимых для расшифрования информации путем перебора всевозможных ключей имеет строгую нижнюю оценку и выходит за пределы возможности современных компьютеров или требует создания использования дорогих вычислительных систем.

6. Незначительное изменение ключа или исходного текста приводит к существенному изменению вида зашифрованного текста.

7. Структурные элементы алгоритма шифрования неизменные.

8. Любой ключ из множества возможных обеспечивает равную криптостойкость.

9. Изменение длины ключа не ведет к качественному ухудшению алгоритма шифрования.

2.1 Стандартный вариант шифрования RC6

Алгоритм RC6 является продолжением криптоалгоритма RC5, разработанного Роном Ривестом (англ. Ron Rivest) – очень известной личностью в мире криптографии. RC5 был незначительно изменен для того, чтобы соответствовать требованиям AES по длине ключа и размеру блока. При этом алгоритм стал еще быстрее, а его ядро, унаследованное от RC5, имеет солидный запас исследований [5. С. 15-20].

Алгоритм является сетью Фейштеля с 4 ветвями смешанного типа: в нем два четных блока используются для одновременного изменения содержимого двух нечетных блоков. Затем производится обычный для сети Фейштеля сдвиг на одно машинное слово, что меняет четные и нечетные блоки местами. Сам алгоритм изображен на рис 1. Разработчики рекомендуют при шифровании использовать 20 раундов сети, хотя в принципе их количество не регламентируется. При 20 повторах операции шифрования алгоритм имеет самую высокую скорость среди 5 финалистов AES.

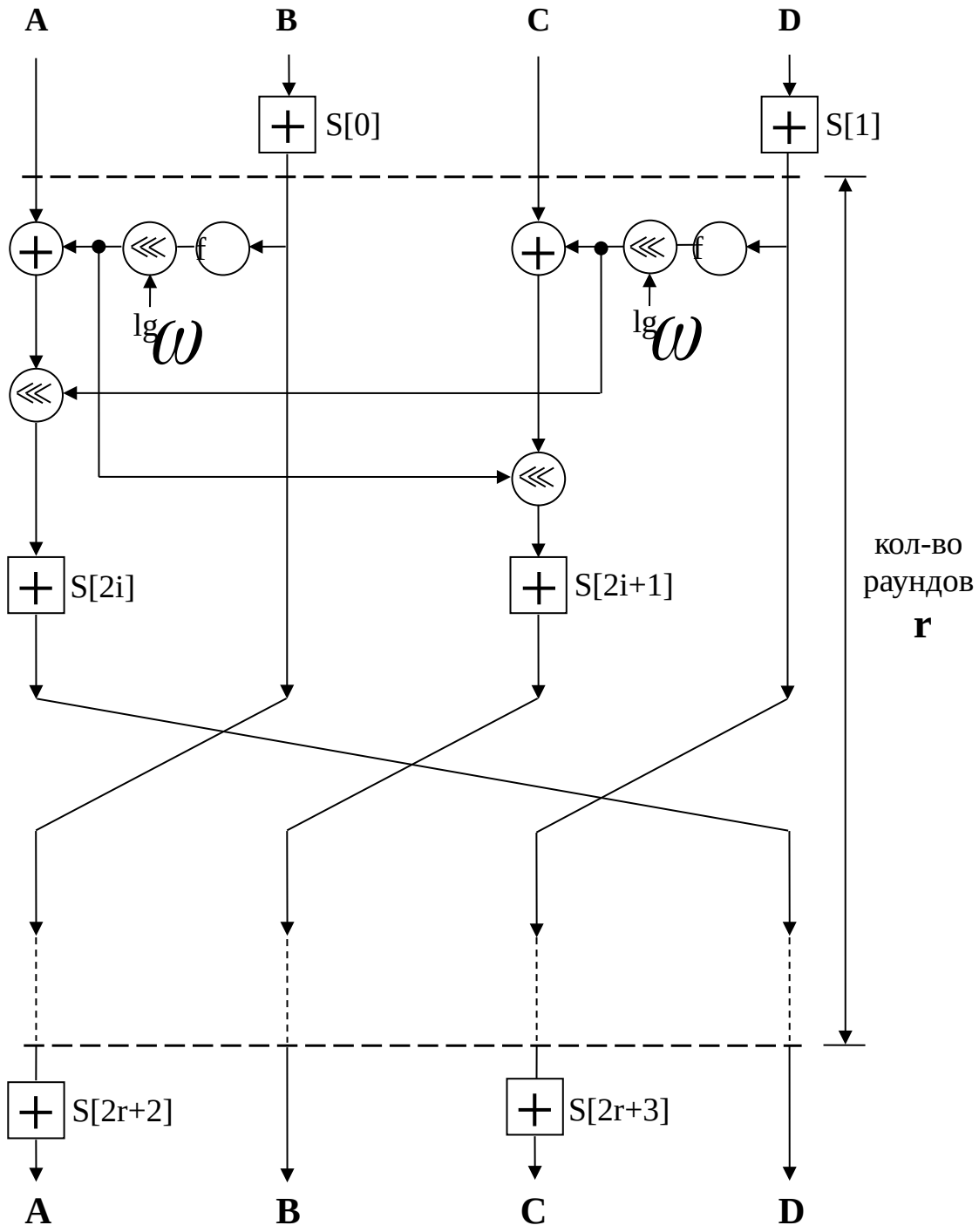
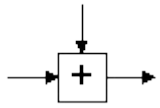
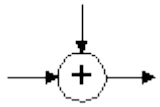
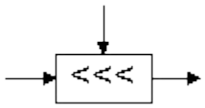
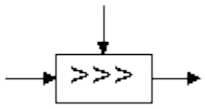


Рис 1. Стандартный алгоритм RC6

Преобразование $f(x)=x*(2*x+1)(\mathrm{mod}\ 2^w)$. Оно используется в качестве нелинейного преобразования с хорошими показателями перемешивания битового значения входной величины. В табл. 1 представлены условные обозначения, применяемые в схематичном рисунке.

Табл. 1. Условные обозначения

Биективные (обратимые) математические функции		
	Сложение	X'=X+V
	Исключающее ИЛИ	X'=X XOR V
Битовые сдвиги		
	Циклический сдвиг влево	X'=X ROL V
	Циклический сдвиг вправо	X'=X ROR V

2.1.1 Процедура шифрования стандартного RC6

Шифрование для RC6-w/r/b (w – длина слова в битах, r – ненулевое количество итерационных циклов зашифрования, b – длина ключа в байтах) описывается следующим образом:

Вход:

Исходный текст, записанный в 4-х w-битовых регистрах A,B,C,D

Число раундов шифрования (перемешивания) регистров r

Ключевая таблица S[0.. 2r + 3] w-битовых слов

Выход:

Шифрованный текст в регистрах A,B,C,D

Процедура:

```
B = B + S[0]
D = D + S[1]

  for i = 1 to r do
  begin
    t = (B * (2 * B + 1)) << log2w
    v = (D * (2 * D + 1)) << log2w
    A = (A xor t) << u + s[2*i]
    C = (C xor u)<< t) + s[2*i+1]
    (A,B,C,D) = (B,C,D,A)
  End
```

```
A = A + S[2*r +2]
C = C + S[2*r +3]
```

Программная реализация алгоритма шифрования:

```
//Шифрование регистров в блоке по алгоритму RC6:
procedure EnCryption;
var
  Temp, A, B, C, D, t, v: Dword;
  i: Byte;
begin
  //Шифрование производится для блока информации в 4-х 32-х
битовых регистрах Buf:
  A := Buf[1];
  B := Buf[2];
  C := Buf[3];
  D := Buf[4];
  B := B + S[0];
  D := D + S[1];
  for i:=1 to r do begin
    t := LRot32(B*(2*B+1), 5);
    v := LRot32(D*(2*D+1), 5);
    A := LRot32(A xor t, v)+S[2*i];
    C := LRot32(C xor v, t)+S[2*i+1];

    Temp := A;
    A := B;
    B := C;
    C := D;
    D := Temp;
    end;

  A := A + S[2*r+2];
  C := C + S[2*r+3];

  PreBuf := Buf;
  Buf[1] := A;
  Buf[2] := B;
  Buf[3] := C;
  Buf[4] := D;
end;
```

Исходный файл разбивается на блоки по 128 бит (16 байт). Эти блоки, в свою очередь, состоят из четырех регистров (которые как раз и используются в четырех ветвях сети Фейштеля). Блоки считываются последовательно в цикле из файла. Первый блок считывается, и к четным регистрам прибавляются по модулю 2^w первые два 32-битовых слова ключа. В цикле над четными регистрами производится операция преобразования $f(x)=x*(2*x+1)(\mathrm{mod}\ 2^w)$ и

циклический сдвиг влево на $\log_2 32 = 5$ бит. Далее над преобразованными регистрами и исходными нечетными регистрами производится операция XOR и циклический сдвиг влево на количество бит, хранимое после преобразования в четных блоках. Заключительная операция в цикле - сложение по модулю 2^w с (2*i)-м и (2*i+1)-м 32-битовыми словами ключа. После окончания цикла к нечетным регистрам прибавляются по модулю 2^w последние два 32-битовых слова ключа. На каждой итерации производится сдвиг на машинное слово, что меняет четные и нечетные регистры местами и начинается новая итерация цикла.

2.1.2 Процедура расшифрования стандартного RC6

Расшифрование выполняется следующим образом:

Вход:

Шифрованный текст, записанный в 4-х w-битовых регистрах A,B,C,D

Число итераций циклов шифрования (перемешивания) регистров r

Ключевая таблица S[0.. 2r + 3] w-битовых слов

Выход:

Исходный текст в регистрах A,B,C,D

Процедура:

```
A = A - S[2*r +2]
C = C - S[2*r +3]

  for i = r downto 1 do
  begin
    (A,B,C,D) = (D,A,B,C)
    t = (B * (2 * B + 1))<< log2w
    v = (D * (2 * D + 1))<< log2w
    C = (C - S[2*i+1]>> t) xor u
    A = (A - S[2*i])>> u) xor t
  end

D = D - S[1]
B = B - S[0]
```

Программная реализация алгоритма расшифрования:

```
//Расшифрование регистров в блоке по алгоритму RC6:
procedure DeCryption;
var
  Temp, A, B, C, D, t, v: Dword;
  i: Byte;
begin
  //Расшифрование производится для блока информации в 4-х 32-х
битовых регистрах Buf:
  A := Buf[1];
  B := Buf[2];
  C := Buf[3];
  D := Buf[4];

  A := A - S[2*r+2];
  C := C - S[2*r+3];

  for i:=r downto 1 do begin
    Temp := D;
    D := C;
    C := B;
    B := A;
    A := Temp;

    t := LRot32(B*(2*B+1), 5);
    v := LRot32(D*(2*D+1), 5);
    A := RRot32(A-S[2*i], v) xor t;
    C := RRot32(C-S[2*i+1], t) xor v;
    end;

  B := B - S[0];
  D := D - S[1];

  PreBuf := Buf;
  Buf[1] := A;
  Buf[2] := B;
  Buf[3] := C;
  Buf[4] := D;
end;
```

Расшифрование аналогично шифрованию.

2.1.3 Процедура генерации ключа стандартного RC6

Алгоритм вычисления ключей для RC6:

Вход:

Определенный пользователем b-байтовый ключ, предварительно загруженный в массив L[0..c-1]

Число раундов шифрования (перемешивания) регистров r

Выход:

Ключевая таблица S[0.. 2r + 3] w-битовых слов

Процедура:

```
S[0] = P32
  for i = 1 to 2*r+3 do
    S[i] = S[i-1] +Q32

v = 3 * max{c,2*r + 4}
A = B = i = j

  for k = 1 to v do
  begin
    A = S[i] = (S[i] + A + B) << 3
    B = L[j] = (L[j] + A + B) <<(AA + BB)
    i = (i+1) mod (2*r+4);
    j = (j+1) mod c;
  end
```

Программная реализация генерации ключа:

```
//Генерация ключа:
procedure Tfmmain.MakeEnKey;
var
  L: array [0..63] of Dword;
  e, k, v, i, j: Byte;
  A, B: Dword;
begin
  Key := fmEncrypt.EdKey.Text;
  Len := Length(Key);

   //Записываем байты в массив из C слов, начиная с младшего,
дописываем в конце нули:
  FillChar(L, Sizeof(L), 0);
  Move(Key, L, Len+1);
  c := Len div 4;
  if Len mod 4 <> 0 then Inc(c);

  //Формирование ключевой таблицы S[0;2r+4] 32-битовых слов
  S[0] := $B7E15163;
  for i:=1 to 2*r+3 do
    S[i] := S[i-1] + $9E3779B9;

  i := 0; j := 0;
  A := 0; B := 0;

  if c>2*r+4 then v := 3*c else v := 3*(2*r+4);
   for k:=1 to v do
   begin
    A := LRot32((S[i]+A+B), 3);
    S[i] := A;
    B := LRot32((L[j]+A+B), (A+B));
```

```
    L[j] := B;
    i := (i+1) mod (2*r+4);
    j := (j+1) mod c;
   end;
end;
```

Алгоритм вычисления ключей для RC6-w/r/b выглядит следующим образом.

Пользователь задает ключ длиной b байтов. Достаточное число ненулевых байтов дописывается в конец, чтобы получилось целое число слов. Затем эти байты записываются, начиная с младшего в массив из слов, т.е. первый байт ключа записывается в L[0], и т.д., а L[c-1] при необходимости дополняется со стороны старших разрядов нулевыми байтами. В результате работы алгоритма генерации ключей будет вычислено 2r + 4 слов, которые будут записаны в массиве ключевой таблицы S[0.. 2r + 3].

Константы P32 = b7e15163 и Q32 = 9e3779b9 – это константы, получаемые из двоичного представления e – 2, где e – основание натурального логарифма, и ф – 1, где ф – золотое сечение соответственно.

После того, как пользователь задает исходный ключ, который записывается в исходный массив L, начинает формироваться ключевая таблица S. Первый элемент S = b7e15163. Далее, в цикле для всех 2*r+3 элементов таблицы S ее элементы задаются следующим образом: каждый следующий элемент равен предыдущему плюс константа 9e3779b9. Далее, определяется, что больше, число ненулевых элементов массива L или 2*r+4. Это значение, помноженное на 3, используется далее в качестве предела итераций для цикла.

В новом цикле снова преобразуются элементы таблицы S. Каждый элемент S[i] равен сумме самого себя, предыдущего элемента S[i-1] и предыдущего элемента L[j-1]. К этому значению применяется циклический сдвиг влево на 3 бит. Аналогичным образом в этом же цикле рассчитывается j-й элемент массива L, за исключение того, что сдвиг влево производится на число бит, равное сумме L[j] и S[i]. Каждые следующие индексы массивов i и j равны самим себе, взятым с увеличением на 1 по модулю (2*r+4) или C соответственно.

2.2 Модифицированный вариант шифрования RC6

В данной работе алгоритм RC6 модифицировался следующим образом:

1. Добавлены 2 новых регистра E и F.

2. Некоторые добавления в генерации ключа.

Длина слова в битах (32 бит) и длина ключа в байтах (256 байт) остались неименными.

Сам модифицированный алгоритм изображен на рис 2.

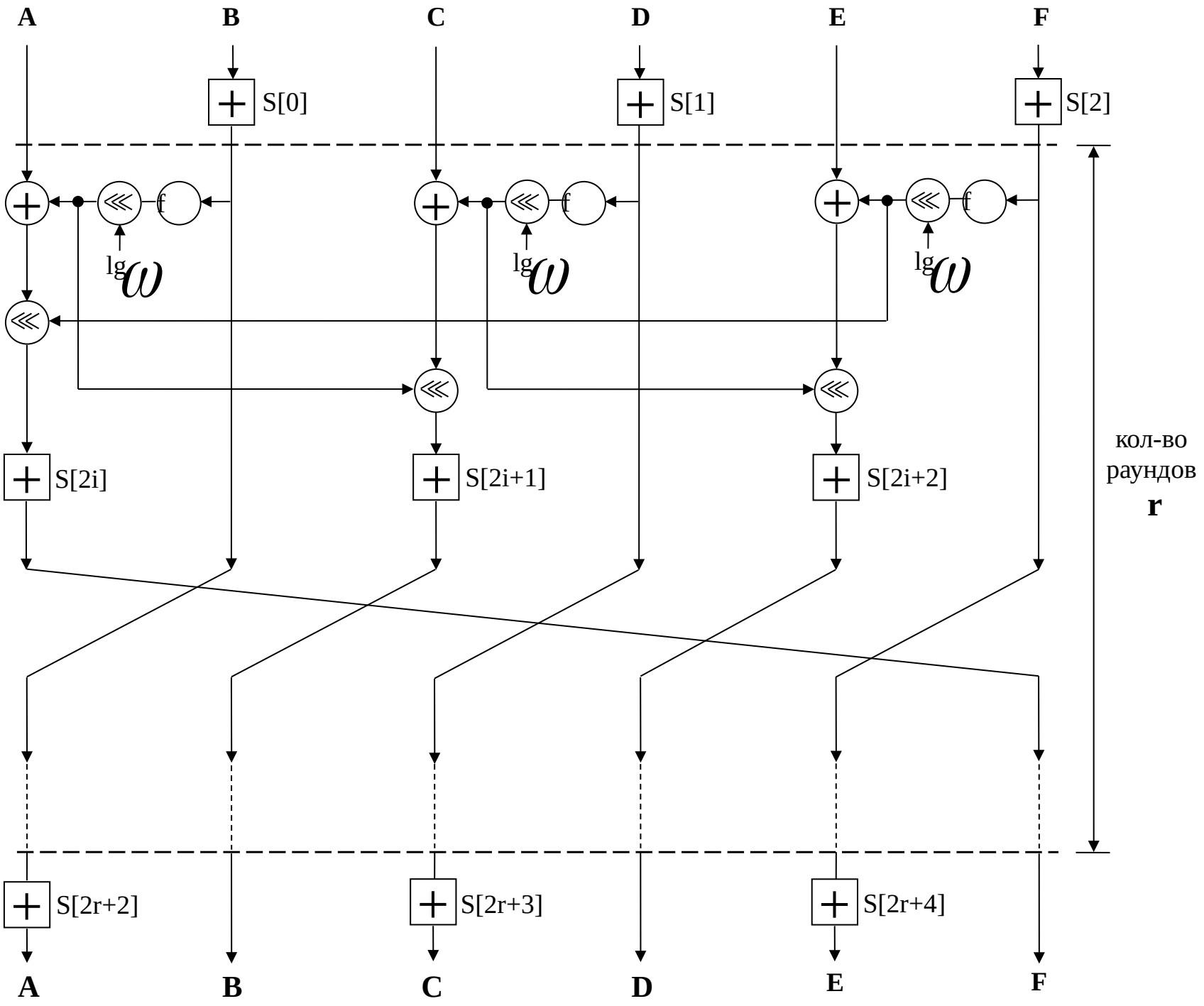

A
B
C
D
E
F
S[0]
S[1]
S[2]
f
lg ω
S[2i]
S[2i+1]
S[2i+2]
кол-во
раундов
r
S[2r+2]
S[2r+3]
S[2r+4]
A
B
C
D
E
F

Рис. 2. Модифицированный алгоритм RC6

2.2.1 Процедура шифрования модифицированного RC6

Шифрование для RC6-w/r/b (w – длина слова в битах, r – ненулевое количество итерационных циклов зашифрования, b – длина ключа в байтах) описывается следующим образом (изменения помечены жирным шрифтом):

Вход:

Исходный текст, записанный в 6-и w-битовых регистрах A,B,C,D,E,F

Число раундов шифрования (перемешивания) регистров r

Ключевая таблица S[0.. 2r + 3] w-битовых слов

Выход:

Шифрованный текст в регистрах A,B,C,D,E,F

Процедура:

```
B = B + S[0];
D = D + S[1];
F = F + S[2];

  for i = 1 to r do
  begin
    t = (B * (2 * B + 1)) << log2w
    u = (D * (2 * D + 1)) << log2w
   x = (F * (2 * F + 1)) << log2w
    A = (A xor t) << x + s[2*i]
    C = (C xor u) << t) + s[2*i+1]
   E = (E xor x) << u) + s[2*i+2]
    (A,B,C,D,E,F) = (F,A,B,C,D,E)
  end

A = A + S[2*r +2]
C = C + S[2*r +3]
E = E + S[2*r +4]
```

Программная реализация алгоритма шифрования:

```
//Шифрование регистров в блоке по алгоритму RC6:
procedure EnCryption;
var
```

```
  Temp, A, B, C, D, E, F, t, v, w: Dword;
  i: Byte;
begin

//Шифрование производится для блока в 6-и 32-х битовых регистрах
Buf:
  A := Buf[1];
  B := Buf[2];
  C := Buf[3];
  D := Buf[4];
  E := Buf[5];
  F := Buf[6];
  B := B + S[0];
  D := D + S[1];
  F := F + S[2];
  for i:=1 to r do
  begin
    t := LRot32(B*(2*B+1), 5);
    v := LRot32(D*(2*D+1), 5);
    w := LRot32(F*(2*F+1), 5);
    A := LRot32(A xor t, w)+S[2*i];
    C := LRot32(C xor v, t)+S[2*i+1];
    E := LRot32(E xor w, v)+S[2*i+2];

    Temp := A;
    A := B;
    B := C;
    C := D;
    D := E;
    E := F;
    F := Temp;
  end;

  A := A + S[2*r+1];
  C := C + S[2*r+2];
  E := E + S[2*r+3];

  PreBuf := Buf;
  Buf[1] := A;
  Buf[2] := B;
  Buf[3] := C;
  Buf[4] := D;
  Buf[5] := E;
  Buf[6] := F;
end;
```

Исходный файл разбивается на блоки по 192 бит (24 байта). Эти блоки, в свою очередь, состоят из шести регистров. Блоки считываются последовательно в цикле из файла. Первый блок считывается, и к четным регистрам прибавляются по модулю 2^w первые два 32-битовых слова ключа. В

цикле над четными регистрами производится операция преобразования $f(x)=x*(2*x+1)(\mathrm{mod}\ 2^w)$ и циклический сдвиг влево на $\log_2 32 = 5$ бит. Далее над преобразованными регистрами и исходными нечетными регистрами производится операция XOR и циклический сдвиг влево на количество бит, хранимое после преобразования в четных блоках. Заключительная операция в цикле - сложение по модулю 2^w с (2*i)-м и (2*i+1)-м 32-битовыми словами ключа. После окончания цикла к нечетным регистрам прибавляются по модулю 2^w последние два 32-битовых слова ключа. На каждой итерации производится сдвиг на машинное слово, что меняет четные и нечетные регистры местами и начинается новая итерация цикла.

2.2.2 Процедура расшифрования модифицированного RC6

Расшифрование выполняется следующим образом (изменения помечены жирным шрифтом):

Вход:

Исходный текст, записанный в 6-и w-битовых регистрах A,B,C,D,E,F

Число раундов шифрования (перемешивания) регистров r

Ключевая таблица S[0.. 3r + 3] w-битовых слов

Выход:

Шифрованный текст в регистрах A,B,C,D,E,F

Процедура:

```
A = A - S[2*r +1]
C = C - S[2*r +2]
E = E – S[2*r +3]

  for i = r downto 1 do
  begin
    (A,B,C,D,E,F) = (F,A,B,C,D,E)
    v = (D * (2 * D + 1))<< log2w
    u = (B * (2 * B + 1))<< log2w
   x = (F * (2 * F + 1))<< log2w
    A = (A - S[2*i])>> x) xor t
    C = (C - S[2*i+1]>> t) xor u
```

```
    E = (E – S[2*i+2]>> u) xor x
   end

D = D - S[1]
B = B - S[0]
F = F – S[2]
```

Программная реализация алгоритма расшифрования:

```
//Расшифрование регистров в блоке по алгоритму RC6:
procedure DeCryption;
var
  Temp, A, B, C, D, E, F, t, v, w: Dword;
  i: Byte;
begin

//Расшифрование производится для блока в 6-и 32-х битовых
регистрах Buf:
  A := Buf[1];
  B := Buf[2];
  C := Buf[3];
  D := Buf[4];
  E := Buf[5];
  F := Buf[6];
  A := A - S[2*r+1];
  C := C - S[2*r+2];
  E := E - S[2*r+3];

  for i:=r downto 1 do
  begin
    Temp := F;
    F := E;
    E := D;
    D := C;
    C := B;
    B := A;
    A := Temp;

    t := LRot32(B*(2*B+1), 5);
    v := LRot32(D*(2*D+1), 5);
    w := LRot32(F*(2*F+1), 5);
    A := RRot32(A-S[2*i], w) xor t;
    C := RRot32(C-S[2*i+1], t) xor v;
    E := RRot32(E-S[2*i+2], v) xor w;
  end;

  B := B - S[0];
  D := D - S[1];
  F := F - S[2];

  PreBuf := Buf;
```

```
  Buf[1] := A;
  Buf[2] := B;
  Buf[3] := C;
  Buf[4] := D;
  Buf[5] := E;
  Buf[6] := F;
end;
```

Расшифрование аналогично шифрованию.

2.2.3 Процедура генерации ключа модифицированного RC6

Алгоритм вычисления ключей для RC6 (изменения помечены жирным шрифтом):

Вход:

Определенный пользователем b-байтовый ключ, предварительно загруженный в массив L[0..e-1]

Число раундов шифрования (перемешивания) регистров r

Выход:

Ключевая таблица S[0.. 2r + 3] w-битовых слов

Процедура:

```
S[0] = P32
  for i = 1 to 2*r+3 do
    S[i] = S[i-1] +Q32

v = 3 * max{e,2*r + 4}
A = B = i = j
C = D = i1 = j1

  for i = 1 to v do
  begin
    A = S[i] = (S[i] + A + B) << 3
    B = L[j] = (L[j] + A + B) <<(AA + BB)
   C = S[i1] = (S[i1] + C + D) >>3
   D = L[j1] = (L[j1) + C + D) >>(CC + DD)
    i = (i+1) mod (2*r+4)
    j = (j+1) mod e
   i1 = (i1+1) mod (2*r+4)
   j1 = (j1+1) mod e
```

```
  end
```

Программная реализация генерации ключа:

```
//Генерация ключа шифрования:
procedure Tfmmain.MakeEnKey;
var
  L: array [0..63] of Dword;
  e, k, v, i, j, i1, j1: Byte;
  A, B, C, D: Dword;
begin
  Key := fmEncrypt.EdKey.Text;
  Len := Length(Key);

   //Записываем байты в массив из C слов, начиная с младшего,
дописываем в конце нули:
  FillChar(L, Sizeof(L), 0);
  Move(Key, L, Len+1);
  e := Len div 4;
  if Len mod 4 <> 0 then Inc(e);

  //Формирование ключевой таблицы S[0;2r+4] 32-битовых слов
  S[0] := $B7E15163;
  for i:=1 to 2*r+3 do
    S[i] := S[i-1] + $9E3779B9;

  i := 0; j := 0; i1 := 0; j1 := 0;
  A := 0; B := 0; C := 0; D := 0;

  if e>2*r+4 then v := 3*e else v := 3*(2*r+4);
   for k:=1 to v do
   begin
    A := LRot32((S[i]+A+B), 3);
    S[i] := A;
    B := LRot32((L[j]+A+B), (A+B));
    L[j] := B;
    i := (i+1) mod (2*r+4);
    j := (j+1) mod e;

    C := RRot32((S[i1]+C+D), 3);
    S[i1] := C;
    D := RRot32((L[j1]+C+D), (C+D));
    L[j1] := D;
    i1 := (i1+1) mod (2*r+4);
    j1 := (j1+1) mod e;
   end;
end;
```

Алгоритм вычисления ключей для RC6-w/r/b выглядит следующим образом.

Пользователь задает ключ длиной b байтов. Достаточное число ненулевых байтов дописывается в конец, чтобы получилось целое число слов. Затем эти байты записываются, начиная с младшего в массив из слов, т.е. первый байт ключа записывается в L[0], и т.д., а L[e-1] при необходимости дополняется со стороны старших разрядов нулевыми байтами. В результате работы алгоритма генерации ключей будет вычислено 2r + 4 слов, которые будут записаны в массиве ключевой таблицы S[0.. 2r + 3].

Константы P32 = b7e15163 и Q32 = 9e3779b9 – это константы, получаемые из двоичного представления e – 2, где e – основание натурального логарифма, и ф – 1, где ф – золотое сечение соответственно.

После того, как пользователь задает исходный ключ, который записывается в исходный массив L, начинает формироваться ключевая таблица S. Первый элемент S = b7e15163. Далее, в цикле для всех 2*r+3 элементов таблицы S ее элементы задаются следующим образом: каждый следующий элемент равен предыдущему плюс константа 9e3779b9. Далее, определяется, что больше, число ненулевых элементов массива L или 2*r+4. Это значение, помноженное на 3, используется далее в качестве предела итераций для цикла.

В новом цикле снова преобразуются элементы таблицы S. Каждый элемент S[i] равен сумме самого себя, предыдущего элемента S[i-1] и предыдущего элемента L[j-1]. К этому значению применяется циклический сдвиг влево на 3 бит. Таким же образом в этом же цикле рассчитывается j-й элемент массива L, за исключение того, что сдвиг влево производится на число бит, равное сумме L[j] и S[i]. Каждые следующие индексы массивов i и j равны самим себе, взятым с увеличением на 1 по модулю (2*r+4) или e соответственно.

Аналогичным образом происходят преобразования для элементов i1 и j1. Каждый элемент S[i1] равен сумме самого себя, предыдущего элемента S[i1-1] и предыдущего элемента L[j1-1]. К этому значению применяется циклический сдвиг вправо на 3 бит. Таким же образом в этом же цикле рассчитывается j1-й элемент массива L, за исключение того, что сдвиг вправо производится на

число бит, равное сумме L[j1] и S[i1]. Каждые следующие индексы массивов i1 и j1 равны самим себе, взятым с увеличением на 1 по модулю (2*r+4) или e соответственно.

В итоге, модифицированный алгоритм RC6 заметно изменился, дополнился новыми преобразованиями: в процедуре шифрования/расшифрования – добавлением 2-х новых регистров E и F, в процедуре генерации ключа – расширением цикла генерации ключа новыми элементами i1 и j1. В связи с этим модифицированный алгоритм стал заметно более криптостойким к криптоанализу, т.к. с добавлением новых элементов его внутренняя структура усложнилась, в ущерб скорости генерации ключа. Однако уменьшилась скорость шифрования и расшифрования, т.к. при добавлении 2-х новых регистров общее количество блоков в шифруемом файле стало на порядок меньше, чем в стандартном алгоритме RC6, размер которых 24 байт и 16 байт соответственно. А для микропроцессорных систем на шифрование одного блока, состоящего из 6-и регистров тратится на пару тактов больше, чем для 4-х регистров. В итоге, общее суммарное количество блоков по 6 регистров в шифруемом файле примерно в 1,5 раза меньше, чем у блоков, размер которых 4 регистра, что можно наблюдать в заметном уменьшении скорости шифрования данных.

Следует отметить, что необходимо различать понятия: количество циклов шифрования (перемешивания) регистров в блоке и количество циклов шифрования блоков в файле. Количество циклов шифрования (перемешивания) регистров в блоке позволяет оперировать только с регистрами (четыре или шесть) в одном блоке, меняет местами четные и нечетные регистры, добавляет к ним генерацию ключа. А количество циклов шифрования блоков в файле позволяет брать отдельный блок (16 или 24 байт) и зашифровывает его опять указанное количество раз. Таких блоков в файле может быть бесчисленное множество, о чем говорит размер шифруемого файла.

Таким образом, была сделана попытка к усовершенствованию существующего алгоритма RC6 до версии RC6.1.

3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ И ИССЛЕДОВАНИЕ КРИПТОСТОЙКОСТИ МОДИФИЦИРОВАННОГО RC6

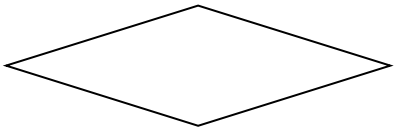
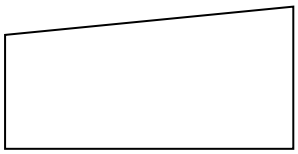
Чтобы понять, каким же образом ведется шифрование/расшифрование файлов, а также подбор пароля, необходимо сначала подробно изучить соответствующие блок-схемы по каждому процессу. К каждой блок-схеме приложено описание. Все блок-схемы справедливы для любого алгоритма RC6, будь-то стандартный или модифицированный. При описании блок-схем в расчет брался модифицированный RC6, где размер блока равен 24 байта, хотя имеет место и стандартный RC6, где размер блока – 16 байт.

В виду того, что RC6 – блочный алгоритм, где происходит шифрование/расшифрование файлов по блокам, то возможно множество вариантов технологии процесса шифрования/расшифрования. В данной работе приведены 2 варианта блок-схем. В программе реализован первый вариант технологии процесса шифрования/расшифрования.

3.1 Блок-схемы процессов шифрующей программы FileCoder на основе RC6

Прежде чем переходить к рассмотрению блок-схем, необходимо, для начала, определиться в их условных обозначениях (Табл. 2)

Табл. 1. Условные обозначения блок-схем

Обозначение	**Блок-схема**
Исходный файл	
Считать	
Цикл	
Условие Если	
Записать в конечный файл	
Записать temp_buff / temp_buff2	
Шифровать / Расшифровать	
Вывод на экран	

3.1.1 Блок-схемы процесса шифрования файла

Блок-схемы процесса шифрования файла представлены в двух вариантах, каждый из которых имеет свои особенности.

3.1.1.1 Первый вариант блок-схемы процесса шифрования файла

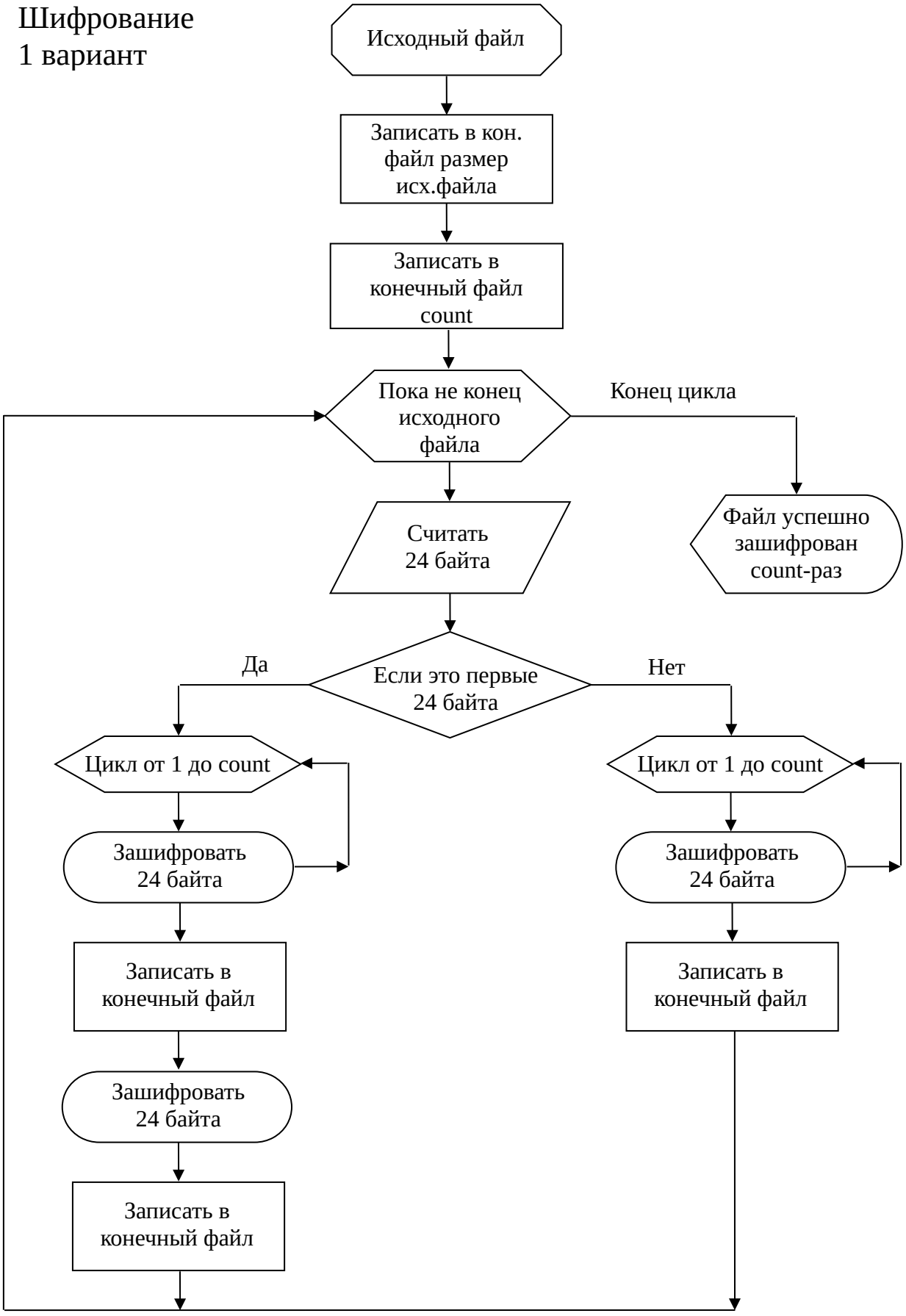
Описание блок-схемы шифрования для первого варианта:

Сначала в конечном файле записываем переменную, содержащую размер исходного незашифрованного файла. Далее, записываем переменную, содержащую количество циклов шифрования исходного файла. В цикле считываем первые 24 байта из исходного файла. Проверяем, если это первые 24 байта, тогда в цикле зашифровываем их указанное количество раз и записываем в конечный файл. Потом еще раз зашифровываем первые 24 байта и записываем в конечный файл. Считываем следующие 24 байта. Проверяем, если это не первые 24 байта, то в цикле зашифровываем их указанное количество раз и записываем в конечный файл. И так далее пока не считаем все блоки в исходном файле. При считывании последнего блока дописываем в его конец нули, если блок окажется менее 24 байт, и также в цикле зашифровываем его указанное количество раз и записываем в конечный файл.

Тем самым итоговый конечный зашифрованный файл содержит в начале 2 4-х байтовых блока. В первый блок входит переменная, содержащая размер исходного незашифрованного файла. Во второй – количество циклов шифрования исходного файла. Третий блок, размер которого 24 байта, зашифрован указанное количество раз, а четвертый зашифрован столько же раз, сколько и третий + еще одно шифрование. Третий и четвертый блоки служат для проверки правильности пароля, введенного при расшифровании, а также они используются в процессе подбора пароля. Все последующие блоки зашифрованы указанное количество раз.

Итого, размер конечного зашифрованного файла на 32 байта больше, чем исходный незашифрованный файл.

Шифрование
1 вариант



Программная реализация процесса шифрования для первого варианта:

```
//Шифрование блоков по алгоритму RC6:
procedure Tfmmain.Coding;
var
  SizeFile, CurrentProgress: integer;
  count, n: integer;
begin
  AssignFile(f1, FileBegin);
  AssignFile(f2, FileEnd);
  Reset(f1, 1);
  Rewrite(f2, 1);
  SizeFile := Filesize(f1);
  count:=strtoint(fmEncrypt.Edit3.Text);
  i := 0;
  BlockWrite(f2, SizeFile, SizeOf(SizeFile), NumWritten);
  BlockWrite(f2, count, SizeOf(count), NumWritten);
  //Пока не конец файла
  repeat
    Inc(i);
    //Если блок <24 байт, то в конец записываются пробелы
    FillChar(Buf, Sizeof(Buf), ord(' '));
    //Считать 24 байта
    BlockRead(f1, Buf, SizeOf(Buf), NumRead);
    if i=1 then
    begin
      for n:=1 to Count do EnCryption;
      BlockWrite(f2, Buf, SizeOf(Buf), NumWritten);
      EnCryption;
      BlockWrite(f2, Buf, SizeOf(Buf), NumWritten);
    end
    else
      begin
        for n:=1 to Count do EnCryption;
        if NumRead <> 0 then BlockWrite(f2,Buf,SizeOf(Buf),
        NumWritten);
      end;
  until (NumRead = 0) Or (SizeOf(Buf) <> NumRead);
  CloseFile(f1);
  CloseFile(f2);
end;
```

3.1.1.2 Второй вариант блок-схемы процесса шифрования файла

Описание блок-схемы шифрования для второго варианта:

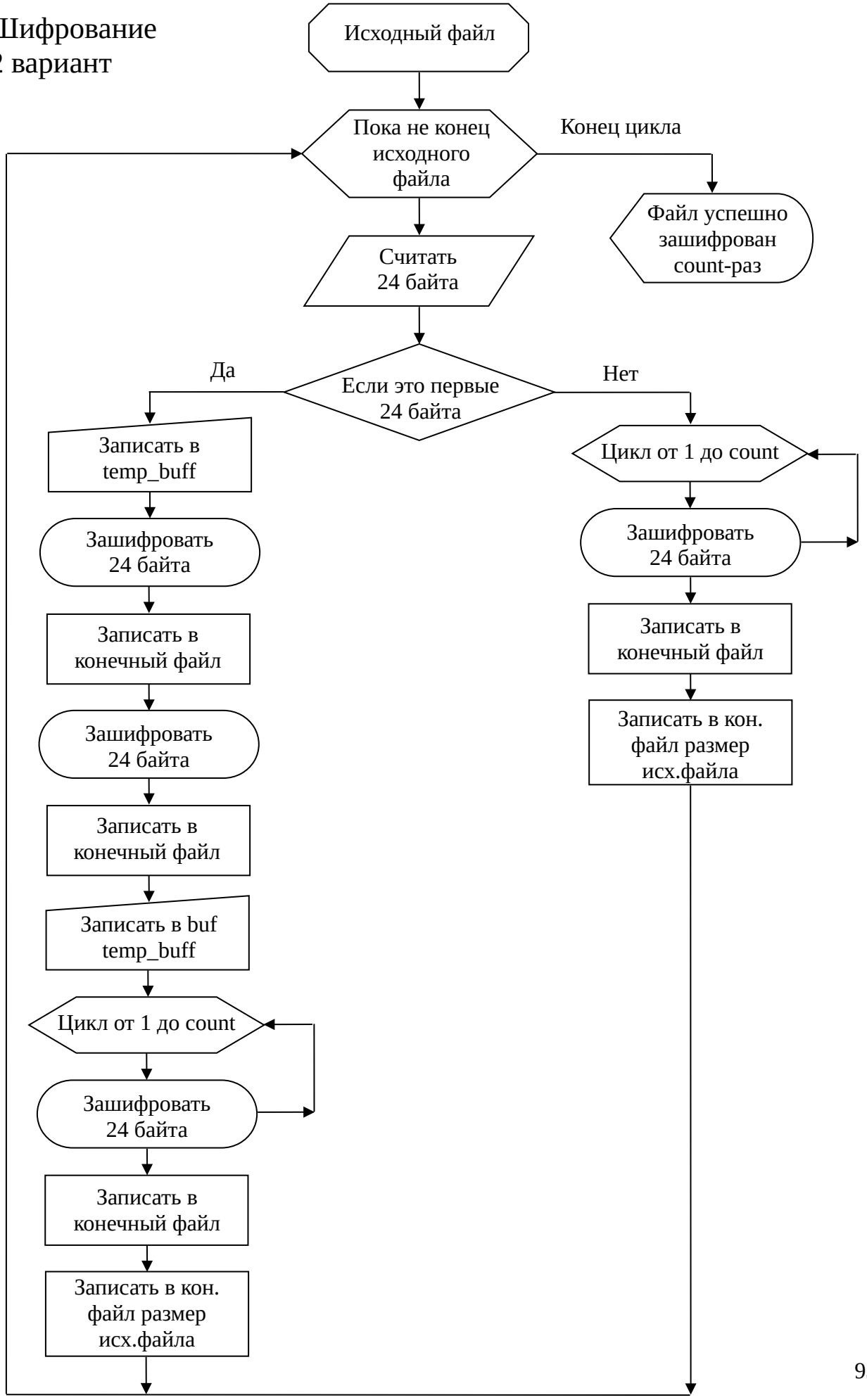
Сначала в цикле считываем первые 24 байта из исходного файла. Проверяем, если это первые 24 байта, тогда записываем их в первую временную переменную. Зашифровываем первые 24 байта и записываем в конечный файл. Потом еще раз зашифровываем первые 24 байта и также

записываем в конечный файл. Из первой временной переменной считываем первые 24 байта, в цикле зашифровываем их указанное количество раз и также записываем в конечный файл. Далее, записываем переменную, содержащую размер исходного незашифрованного файла. Считываем следующие 24 байта. Проверяем, если это не первые 24 байта, то в цикле зашифровываем их указанное количество раз и записываем в конечный файл. И так далее пока не считаем все блоки в исходном файле. При считывании последнего блока дописываем в его конец нули, если блок окажется менее 24 байт, и также в цикле зашифровываем его указанное количество раз и записываем в конечный файл.

Тем самым в итоговом конечном зашифрованном файле первый блок зашифрован 1 раз, второй – 2 раза. Эти два блока служат для проверки правильности пароля, введенного при расшифровании, а также используются в процессе подбора пароля. Третий блок зашифрован указанное количество раз. Четвертый блок, размер которого 4 байта, содержит размер исходного незашифрованного файла. Все последующие блоки зашифрованы указанное количество раз.

Итого, размер конечного зашифрованного файла на 52 байта больше, чем исходный незашифрованный файл.

Шифрование
2 вариант



Программная реализация процесса шифрования для второго варианта:

```
//Шифрование блоков по алгоритму RC6:
procedure Tfmmain.Coding;
var
  SizeFile, CurrentProgress: integer;
  count, n: integer;
begin
  AssignFile(f1, FileBegin);
  AssignFile(f2, FileEnd);
  Reset(f1, 1);
  Rewrite(f2, 1);
  SizeFile := Filesize(f1);
  count:=strtoint(fmEncrypt.Edit3.Text);
  i := 0;
  //Пока не конец файла
  repeat
    Inc(i);
    //Если блок <24 байт, то в конец записываются пробелы
    FillChar(Buf, Sizeof(Buf), ord(' '));
    //Считать 24 байта
    BlockRead(f1, Buf, SizeOf(Buf), NumRead);
    if i=1 then
    begin
      Temp_Buff:=Buf;
      EnCryption;
      BlockWrite(f2, Buf, SizeOf(Buf), NumWritten);
      EnCryption;
      BlockWrite(f2, Buf, SizeOf(Buf), NumWritten);
      Buf:=Temp_Buff;
      for n:=1 to Count do EnCryption;
      BlockWrite(f2, Buf, SizeOf(Buf), NumWritten);
      //Записать в файл SizeFile
      BlockWrite(f2, SizeFile, SizeOf(SizeFile));
    end
    else
      begin
        for n:=1 to Count do EnCryption;
      if NumRead <> 0 then BlockWrite(f2, Buf, SizeOf(Buf),
      NumWritten);
      end;
  until (NumRead = 0) Or (SizeOf(Buf) <> NumRead);
  CloseFile(f1);
  CloseFile(f2);
end;
```

Исходя из представленных двух вариантах шифрования файла можно сделать вывод, что второй вариант более устойчив к взлому, т.к. не содержит в своем файле переменную, указывающую сколько раз данный файл был зашифрован. Скорость шифрования файлов одинакового размера примерно

равны у обоих вариантов процесса шифрования файла. При использовании первого варианта конечный файл на 32 байта больше исходного, а при использовании второго – на 52 байта.

3.1.2 Блок-схемы процесса расшифрования файла

Блок-схемы процесса расшифрования файла представлены в двух вариантах, каждый из которых имеет свои особенности.

3.1.2.1 Первый вариант блок-схемы процесса расшифрования файла

Описание блок-схемы расшифрования для первого варианта:

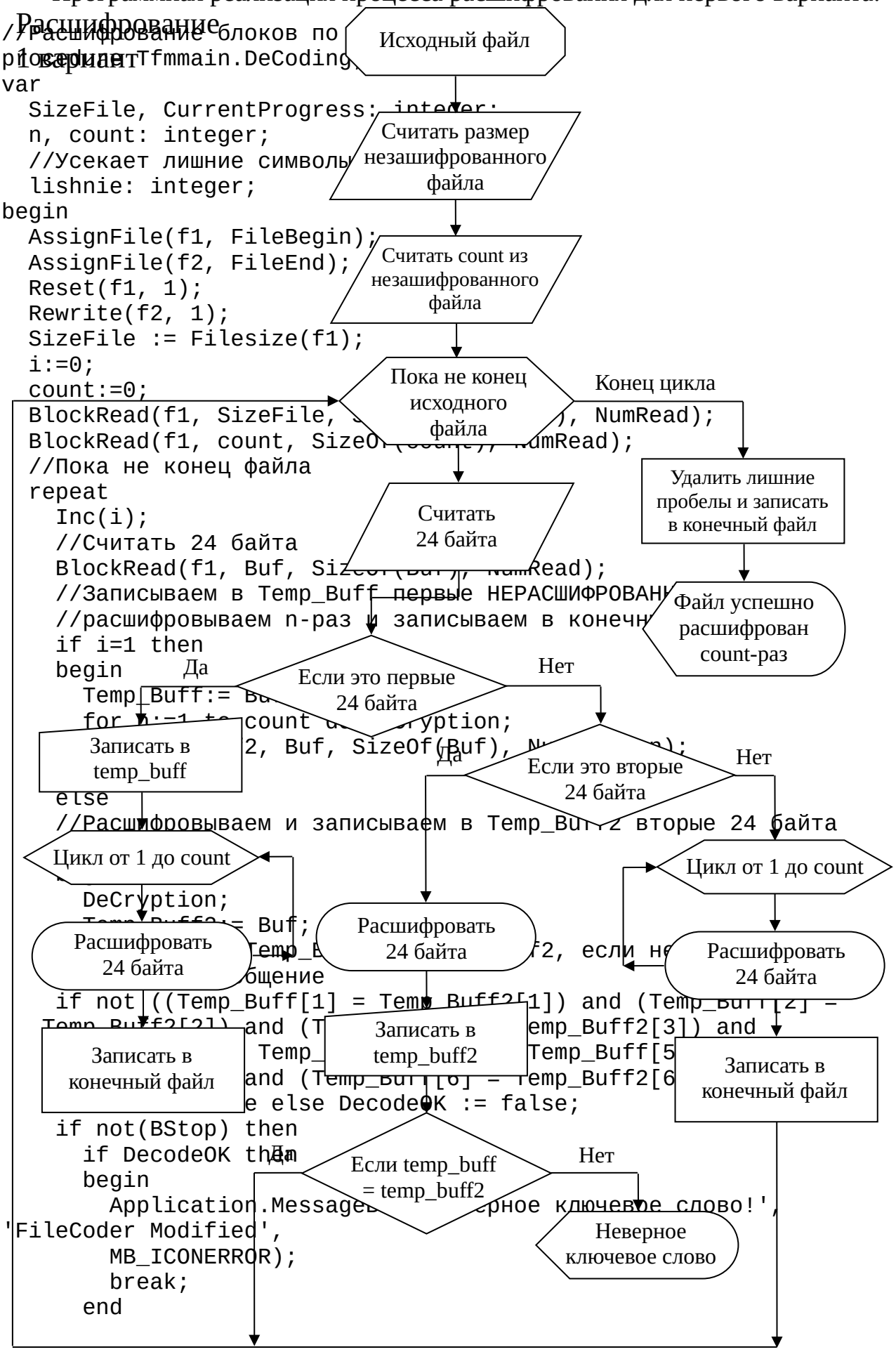
Сначала считываем и запоминаем в переменной первые 4 байта в исходном зашифрованном файле, которые содержат размер исходного незашифрованного файла. Далее, считываем и запоминаем в переменной следующие 4 байта, содержащие количество циклов шифрования исходного незашифрованного файла. На следующем этапе в цикле считываем 24 байта из исходного файла. Проверяем, если это первые 24 байта, тогда сразу записываем их в первую временную переменную. В цикле расшифровываем первые 24 байта указанное количество раз при шифровании исходного незашифрованного файла и записываем в конечный файл. Считываем следующие 24 байта. Проверяем, если это вторые 24 байта, то расшифровываем их и записываем во вторую временную переменную. Сравниваем равны ли первые 24 байта из первой временной переменной со вторыми 24 байтами из второй временной переменной. Если не равны, то выводим на экран сообщение о неверно набранном пароле. Считываем следующие 24 байта. Проверяем, если это не первые и не вторые 24 байта, то в цикле расшифровываем их указанное количество раз при шифровании исходного незашифрованного файла и записываем в конечный файл. На последнем этапе расшифрования удаляем лишние пробелы из конечного расшифрованного файла, тем самым его размер стал равен исходному незашифрованному файлу.

Тем самым итоговый конечный расшифрованный файл стал на 32 байт меньше, которые содержали контрольные блоки для размера и количества циклов шифрования исходного незашифрованного файла, а также контрольный блок проверки для информирования пользователя о том, правильно ли он ввел пароль при расшифровании файла или нет.

Программная реализация процесса расшифрования для первого варианта:

Расшифрование
1 вариант

```
//Расшифрование блоков по
procedure Tfmmain.DeCoding;
var
  SizeFile, CurrentProgress: integer;
  n, count: integer;
  //Усекает лишние символы
  lishnie: integer;
begin
  AssignFile(f1, FileBegin);
  AssignFile(f2, FileEnd);
  Reset(f1, 1);
  Rewrite(f2, 1);
  SizeFile := Filesize(f1);
  i:=0;
  count:=0;
  BlockRead(f1, SizeFile, S                ), NumRead);
  BlockRead(f1, count, SizeOf(count), NumRead);
  //Пока не конец файла
  repeat
    Inc(i);
    //Считать 24 байта
    BlockRead(f1, Buf, SizeOf(Buf), NumRead);
    //Записываем в Temp_Buff первые НЕРАСШИФРОВАН
    //расшифровываем n-раз и записываем в конечн
    if i=1 then
    begin
      Temp_Buff:= Bu
      for n:=1 to count d      ryption;
                2, Buf, SizeOf(Buf), N        n);
    else
    //Расшифровываем и записываем в Temp_Buff2 вторые 24 байта
    f
      DeCryption;
      Temp_Buff2:= Buf;
                Temp_B                f2, если не
                бщение
    if not ((Temp_Buff[1] = Temp_Buff2[1]) and (Temp_Buff[2] =
  Temp_Buff2[2]) and (T                emp_Buff2[3]) and
                Temp_                Temp_Buff[5
                and (Temp_Buff[6] = Temp_Buff2[6
                e else DecodeOK := false;
    if not(BStop) then
      if DecodeOK then
      begin
        Application.Message                ерное ключевое слово!',
'FileCoder Modified',
        MB_ICONERROR);
        break;
      end
```



```
    end
    else
      begin
        for n:=1 to count do DeCryption;
        if NumRead <> 0 then BlockWrite(f2, Buf, SizeOf(Buf),
        NumWritten);
      end;
  until (NumRead = 0) Or (SizeOf(Buf) <> NumRead);
  lishnie := Filesize(f2) - SizeFile;
  Seek(f2, Filesize(f2) - lishnie);
  Truncate(f2);
  CloseFile(f1);
  CloseFile(f2);
end;
```

3.1.2.2 Второй вариант блок-схемы процесса расшифрования файла

Описание блок-схемы расшифрования для второго варианта:

Сначала в цикле считываем 24 байта из исходного файла. Проверяем, если это первые 24 байта, тогда сразу записываем их в первую временную переменную. Далее, считываем следующие 24 байта. Проверяем, если это вторые 24 байта, то расшифровываем их и записываем во вторую временную переменную. Проверяем, равны ли первые 24 байта из первой временной переменной со вторыми 24 байтами из второй временной переменной. Если не равны, то выводим на экран сообщение о неверно набранном пароле. Считываем следующие 24 байта. Проверяем, если это третьи 24 байта, то пока условие истинно, увеличиваем на единицу счетчик и записываем во вторую временную переменную третьи 24 байта. Сравниваем, если первые 24 байта из первой временной переменной равны с третьими 24 байтами из второй временной переменной, то расшифровываем третьи 24 байта и записываем в конечный файл. Если они не равны, то из второй временной переменной считываем третьи 24 байта, расшифровываем их и продолжаем цикл с истинным условием. Потом сразу считываем и запоминаем в переменной размер исходного незашифрованного файла. Считываем следующие 24 байта. Проверяем, если это не первые 24 байта, то в цикле зашифровываем их указанное количество раз и записываем в конечный файл. И так далее пока не считаем все блоки в исходном файле. На последнем этапе расшифрования

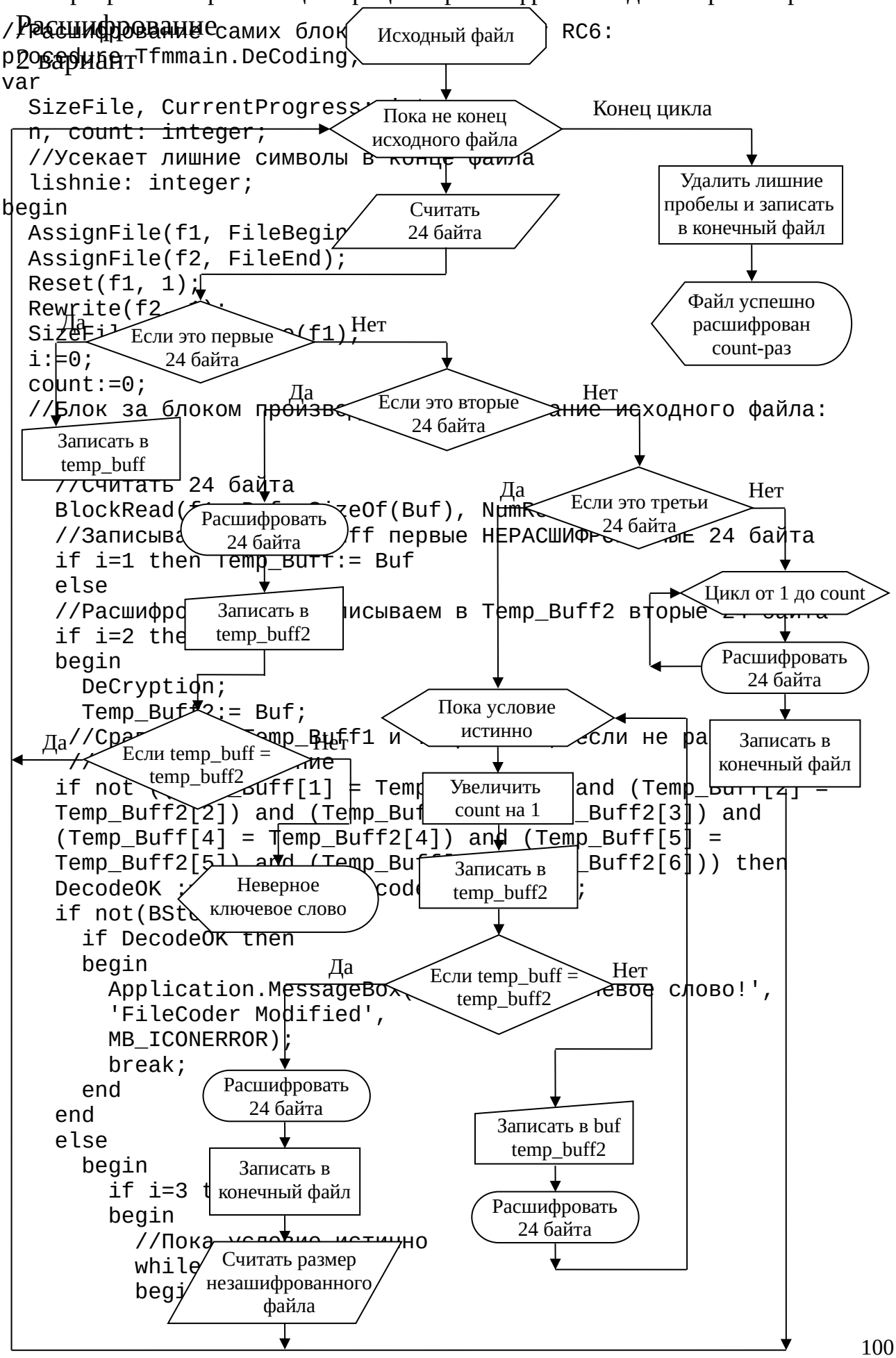
удаляем лишние пробелы из конечного расшифрованного файла, тем самым его размер стал равен исходному незашифрованному файлу.

Тем самым итоговый конечный расшифрованный файл стал на 52 байт меньше, которые содержали контрольный блок для размера исходного незашифрованного файла, а также контрольные блоки проверки для информирования пользователя о том, правильно ли он ввел пароль при расшифровании файла или нет.

Программная реализация процесса расшифрования для второго варианта:

Расшифрование
2 вариант

```
//Расшифрование самих блок                    RC6:
procedure Tfmmain.DeCoding;
var
  SizeFile, CurrentProgress:
  n, count: integer;
  //Усекает лишние символы в конце файла
  lishnie: integer;
begin
  AssignFile(f1, FileBegin
  AssignFile(f2, FileEnd);
  Reset(f1, 1);
  Rewrite(f2
  SizeFil                (f1);
  i:=0;
  count:=0;
  //Блок за блоком произв                ание исходного файла:
    //Считать 24 байта
    BlockRead(f                  zeOf(Buf), NumR
    //Записыва                ff первые НЕРАСШИФР                ЫЕ 24 байта
    if i=1 then Temp_Buff:= Buf
    else
    //Расшифро                исываем в Temp_Buff2 вторые 24 байта
    if i=2 the
    begin
      DeCryption;
      Temp_Buff2:= Buf;
      //Сра                emp_Buff1 и                если не ра
      //                ние
    if not (                Buff[1] = Temp                and (Temp_Buff[2] =
    Temp_Buff2[2]) and (Temp_Buf                _Buff2[3]) and
    (Temp_Buff[4] = Temp_Buff2[4]) and (Temp_Buff[5] =
    Temp_Buff2[5]) and (Temp_Buf                _Buff2[6])) then
    DecodeOK :                code                ;
    if not(BSt
      if DecodeOK then
      begin
        Application.MessageBox(                    евое слово!',
        'FileCoder Modified',
        MB_ICONERROR);
        break;
      end
    end
    else
      begin
        if i=3 t
        begin
          //Пока условие истинно
          while
          begi
```



```
            inc(count);
            Temp_Buff2:=Buf;
            if  ((Temp_Buff[1] = Temp_Buff2[1]) and (Temp_Buff[2]
            = Temp_Buff2[2])
            and (Temp_Buff[3] = Temp_Buff2[3]) and (Temp_Buff[4] =
            Temp_Buff2[4]) and (Temp_Buff[5] = Temp_Buff2[5]) and
            (Temp_Buff[6] = Temp_Buff2[6]))
            then break;
            Buf:=Temp_Buff2;
            DeCryption;
          end;
        DeCryption;
        BlockWrite(f2, Buf, SizeOf(Buf), NumWritten);
        //Считываем SizeFile
        BlockRead(f1, SizeFile, SizeOf(SizeFile));
         end
          else
          begin
            for n:=1 to count do DeCryption;
            if NumRead <> 0 then BlockWrite(f2, Buf, SizeOf(Buf),
NumWritten);
          end;
      end;
  until (NumRead = 0) Or (SizeOf(Buf) <> NumRead);
  lishnie:=Filesize(f2)-SizeFile;
  Seek(f2, Filesize(f2)-lishnie);
  Truncate(f2);
  CloseFile(f1);
  CloseFile(f2);
end;
```

В результате, блок-схемы процесса расшифрования довольно внушительны у обоих вариантов. Однако это не мешает программе также быстро и безошибочно расшифровывать файлы и делать их того же размера, что и исходные файлы.

3.1.3 Блок-схемы процесса подбора пароля

Блок-схемы процесса подбора пароля представлены в двух вариантах, которые очень похожи друг на друга. Разница лишь в позиционировании указателя в файле.

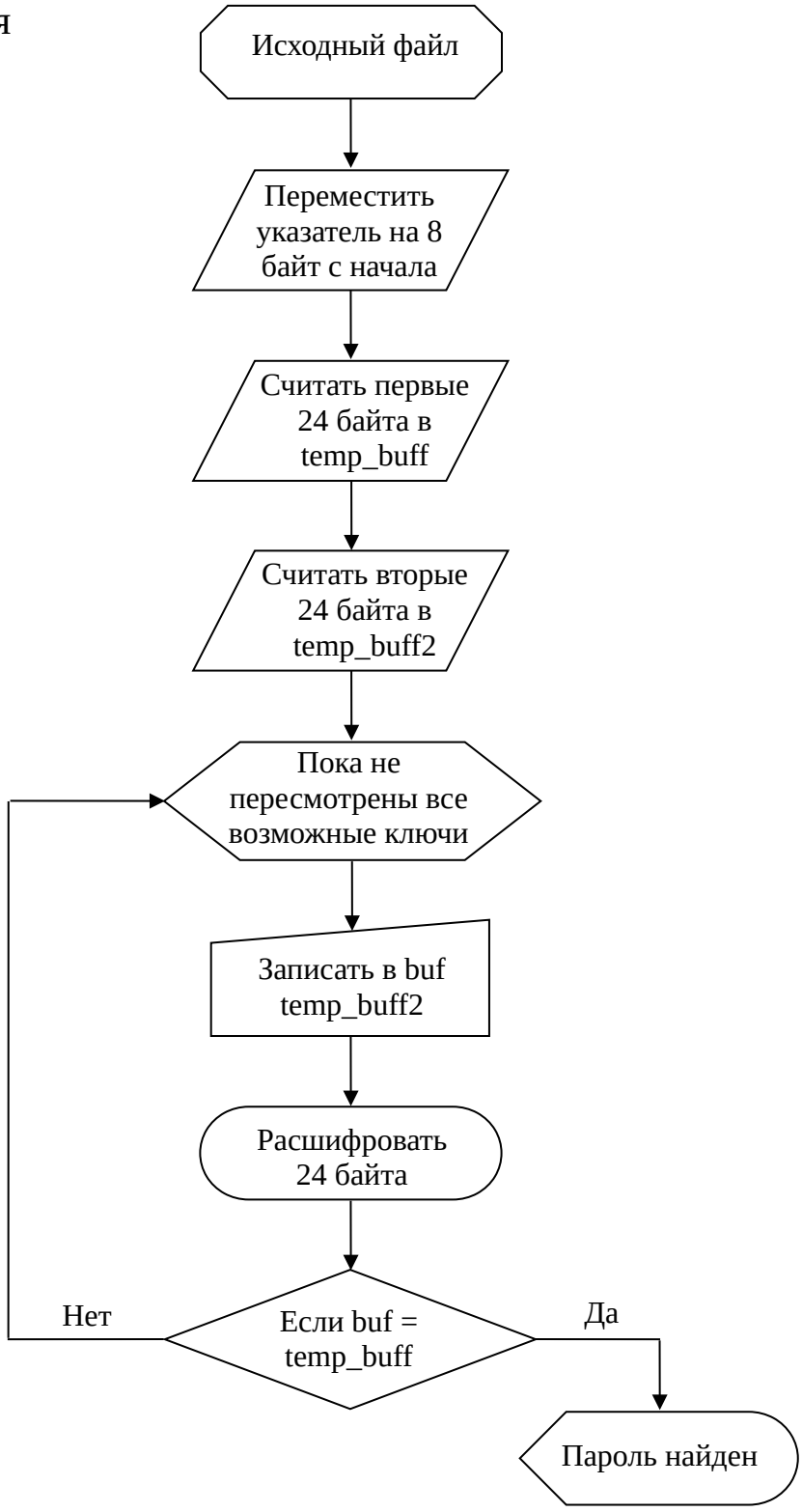
3.1.3.1 Первый вариант блок-схемы процесса подбора пароля

Описание блок-схемы подбора пароля для первого варианта:

Открываем исходный файл и перемещаем в нем указатель на 8-ю позицию. Считываем первые 24 байта в первую временную переменную. Считываем вторые 24 байта во вторую временную переменную. В цикле пока не пересмотрены все возможные ключи, считываем вторые 24 байта из второй временной переменной и расшифровываем их. Сравниваем, если вторые 24 байта равны первым 24 байтам из первой временной переменной, тогда выводим сообщение, что пароль найден. В противном случае, продолжаем пересматривать все возможные ключи из указанного диапазона.

Тем самым поиск нужного пароля ведется только по первому и расшифрованному второму блоках в файле.

Подбор пароля
1 вариант



Программная реализация процесса подбора пароля для первого варианта:

```
//Подбор пароля
procedure Tfmmain.FindPassword;
var
  //переменные генерации ключа
  L: array [0..63] of Dword;
  e, k, v, i, j, i1, j1: Byte;
  A, B, C, D: Dword;

  //переменные поиска по цифрам
  CurrentKey: string;
  CurFile: TFileStream;
begin
  CurrentKey := '';

  //Запоминаем начальное время
  CurFile:= TFileStream.Create(FileBegin, fmOpenRead);
  CurFile.Seek(8, soFromBeginning);
  //Считываем в Temp_Buff первые 24 байта из файла
  CurFile.Read(Temp_Buff, SizeOf(Temp_Buff));
  //Передвигаем указатель на вторые 24 байта
  CurFile.Seek(32, soFromBeginning);
  //Считываем в Temp_Buff2 вторые 24 байта из файла
  CurFile.Read(Temp_Buff2, SizeOf(Temp_Buff2));
  CurFile.Destroy;
  //цикл перебора паролей
  while true do
  begin
    if fmPassword.rbNumber.Checked then
       CurrentKey:= DigitalPassword(CurrentKey);
    if fmPassword.rbWord.Checked then
       CurrentKey:= WordPassword(CurrentKey);
    if fmPassword.rbWordsDigits.Checked then
       CurrentKey:= WordsDigits(CurrentKey);
    if fmPassword.rbSpisok.Checked then
       CurrentKey:=
SpisokPassword(CurrentKey,fmPassword.edSpisok.Text);
    if fmPassword.rbAll.Checked then
       CurrentKey:= AllPassword(CurrentKey);
    if fmPassword.cbShowPass.Checked then
      fmPassword.lbCurrPass.Caption:=CurrentKey;

  //Генерация ключа расшифрования
  Key := CurrentKey;
  Len := Length(Key);

  //Генерация переменных ключа
  FillChar(L, Sizeof(L), 0);
  Move(Key, L, Len+1);
  e := Len div 4;
  if Len mod 4 <> 0 then Inc(e);
```

```
  //Формирование ключевой таблицы S[0;2r+4] 32-битовых слов
  S[0] := $B7E15163;
  for i:=1 to 2*r+3 do
    S[i] := S[i-1] + $9E3779B9;

  i := 0; j := 0; i1 := 0; j1 := 0;
  A := 0; B := 0; C := 0; D := 0;

  if e>2*r+4 then v := 3*e else v := 3*(2*r+4);
   for k:=1 to v do
   begin
    A := LRot32((S[i]+A+B), 3);
    S[i] := A;
    B := LRot32((L[j]+A+B), (A+B));
    L[j] := B;
    i := (i+1) mod (2*r+4);
    j := (j+1) mod e;

    C := RRot32((S[i1]+C+D), 3);
    S[i1] := C;
    D := RRot32((L[j1]+C+D), (C+D));
    L[j1] := D;
    i1 := (i1+1) mod (2*r+4);
    j1 := (j1+1) mod e;
   end;

   //Расшифрование вторых 24 байт
   Buf := Temp_Buff2;
   DeCryption;
  //если Buff = Temp_Buff, значит пароль найден, останавливаем
цикл, выводим пароль
    if (Temp_Buff[1] = Buf[1]) and (Temp_Buff[2] = Buf[2]) and
    (Temp_Buff[3] = Buf[3]) and (Temp_Buff[4] = Buf[4]) and
    (Temp_Buff[5] = Buf[5]) and (Temp_Buff[6] = Buf[6]) then
     break;
  end;
end;
```

3.1.3.2 Второй вариант блок-схемы процесса подбора пароля

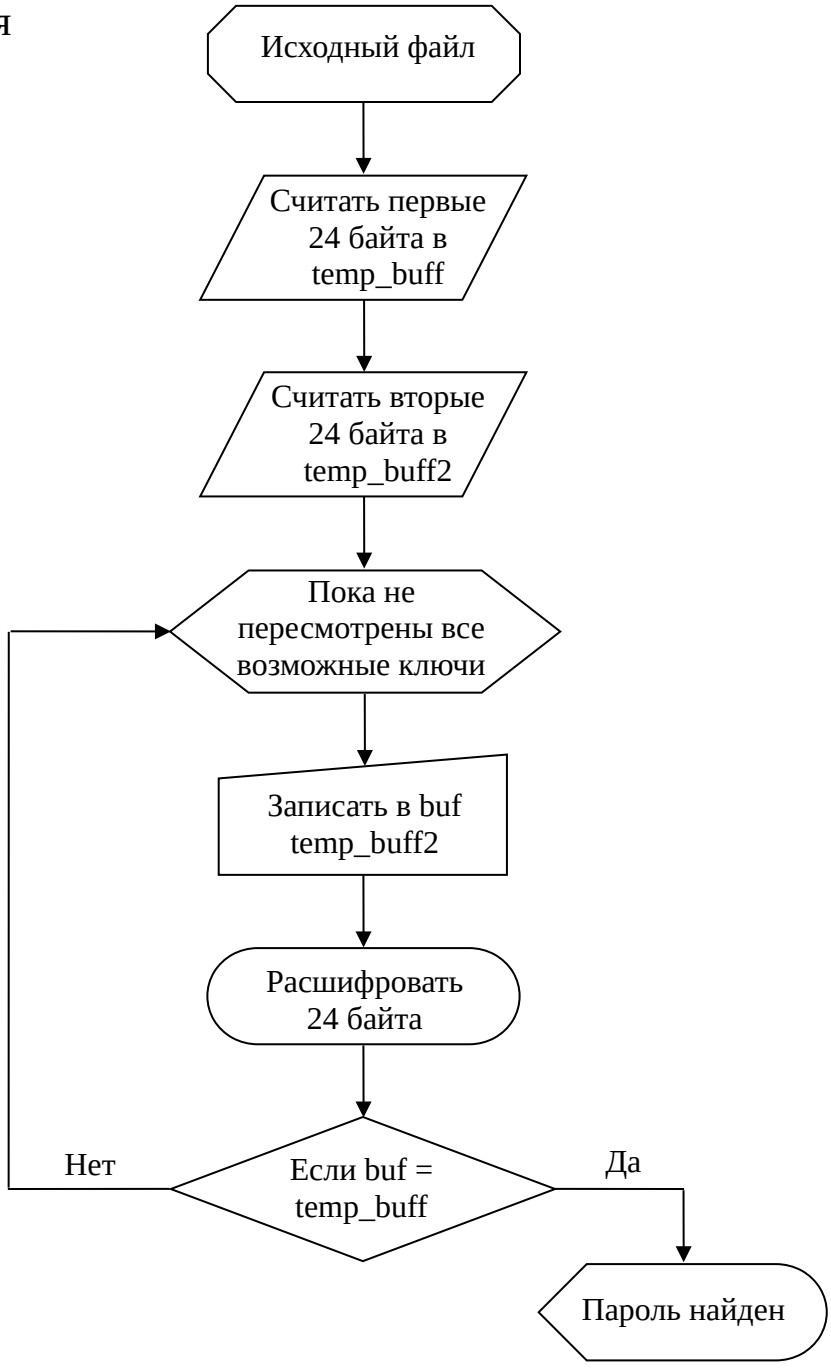
Описание блок-схемы подбора пароля для второго варианта:

В исходном файле считываем первые 24 байта в первую временную переменную. Считываем вторые 24 байта во вторую временную переменную. В цикле пока не пересмотрены все возможные ключи, считываем вторые 24 байта из второй временной переменной и расшифровываем их. Сравниваем, если вторые 24 байта равны первым 24 байтам из первой временной переменной,

тогда выводим сообщение, что пароль найден. В противном случае, продолжаем пересматривать все возможные ключи из указанного диапазона.

Тем самым поиск нужного пароля ведется только по первому и расшифрованному второму блоках в файле.

Подбор пароля
2 вариант



Программная реализация процесса подбора пароля для второго варианта:

```
//Подбор пароля
procedure Tfmmain.FindPassword;
var
  //переменные генерации ключа
  L: array [0..63] of Dword;
  e, k, v, i, j, i1, j1: Byte;
  A, B, C, D: Dword;

  //переменные поиска по цифрам
  CurrentKey: string;
  CurFile: TFileStream;
begin
  CurrentKey := '';
  CurFile:= TFileStream.Create(FileBegin, fmOpenRead);
  //Считываем в Temp_Buff первые 24 байта из файла
  CurFile.Read(Temp_Buff, SizeOf(Temp_Buff));
  //Передвигаем указатель на вторые 24 байта
  CurFile.Seek(24, soFromBeginning);
  //Считываем в Temp_Buff2 вторые 24 байта из файла
  CurFile.Read(Temp_Buff2, SizeOf(Temp_Buff2));
  CurFile.Destroy;
  //цикл перебора паролей
  while true do
  begin
    if fmPassword.rbNumber.Checked then
       CurrentKey:= DigitalPassword(CurrentKey);
    if fmPassword.rbWord.Checked then
       CurrentKey:= WordPassword(CurrentKey);
    if fmPassword.rbWordsDigits.Checked then
       CurrentKey:= WordsDigits(CurrentKey);
    if fmPassword.rbSpisok.Checked then
                                                    CurrentKey:=
SpisokPassword(CurrentKey,fmPassword.edSpisok.Text);
    if fmPassword.rbAll.Checked then
       CurrentKey:= AllPassword(CurrentKey);
    if fmPassword.cbShowPass.Checked then
      fmPassword.lbCurrPass.Caption:=CurrentKey;

  //Генерация ключа расшифрования
  Key := CurrentKey;
  Len := Length(Key);

  //Генерация переменных ключа
  FillChar(L, Sizeof(L), 0);
  Move(Key, L, Len+1);
  e := Len div 4;
  if Len mod 4 <> 0 then Inc(e);

  //Формирование ключевой таблицы S[0;2r+4] 32-битовых слов
  S[0] := $B7E15163;
  for i:=1 to 2*r+3 do
```

```
    S[i] := S[i-1] + $9E3779B9;

  i := 0; j := 0; i1 := 0; j1 := 0;
  A := 0; B := 0; C := 0; D := 0;

  if e>2*r+4 then v := 3*e else v := 3*(2*r+4);
   for k:=1 to v do
   begin
    A := LRot32((S[i]+A+B), 3);
    S[i] := A;
    B := LRot32((L[j]+A+B), (A+B));
    L[j] := B;
    i := (i+1) mod (2*r+4);
    j := (j+1) mod e;

    C := RRot32((S[i1]+C+D), 3);
    S[i1] := C;
    D := RRot32((L[j1]+C+D), (C+D));
    L[j1] := D;
    i1 := (i1+1) mod (2*r+4);
    j1 := (j1+1) mod e;
   end;

   //Расшифрование вторых 24 байт
   Buf := Temp_Buff2;
   DeCryption;
//если Buff = Temp_Buff, значит пароль найден, останавливаем цикл,
// выводим пароль
    if (Temp_Buff[1] = Buf[1]) and (Temp_Buff[2] = Buf[2]) and
    (Temp_Buff[3] = Buf[3]) and (Temp_Buff[4] = Buf[4]) and
    (Temp_Buff[5] = Buf[5]) and (Temp_Buff[6] = Buf[6]) then
    break;
  end;
end;
```

3.2 Тестирование и исследование технологии шифрующей программы FileCoder на основе RC6

В результате работы была разработана программа FileCoder шифрования/расшифрования данных по алгоритму RC6 со следующими параметрами: длина слова в битах (w) = 32, количество циклов шифрования (r) = 20 (в стандартном варианте) и 40 (в модифицированном варианте), размер ключа в байтах (b) = 1..255. Ключом может быть любая как буквенно-цифровая последовательность символов латинского и русского алфавита, так и другие символы из ASCII таблицы. Для расшифрования применяется тот же самый

ключ, который был использован в шифровании. Программа была написаны на языке программирования Borland Delphi 7.0. Для реализации циклических сдвигов был использован язык Assembler, интегрированный в язык Object Pascal.

Тестирование шифрующей программы FileCoder на основе RC6 выполнялось на компьютере с процессором Pentium IV 2,8 ГГц с ОС Windows XP SP2.

3.2.1 Шифрование/расшифрование файлов

При исследовании шифрования/расшифрования файла был использован один и тот же ключ. Следует отметить, что расширение файла и длина его ключа не влияют на время шифрования/расшифрования.

В табл. 2 указаны примерный размер файлов в Мб и время в час, мин и сек.

Табл. 2. Размер файлов и время его шифрования/расшифрования

Вариант программы	**~11,9 Мб**	**~48 Мб**	**~97,6 Мб**	**~214 Мб**	**~850 Мб**	**~1940 Мб**
FileCoder Standart	18с	1м 16с	2м 35с	5м 42с	23м 05с	1ч 10м 25с
FileCoder Modified	13с	52с	1м 46с	3м 58с	16м 25с	45м 21с

Результаты тестирования:

1. Модифицированный алгоритм RC6 на много быстрее зашифровывает и расшифровывает файлы. Это связано с тем, что в нем размер блока равен 24 байта, и следовательно, общее количество блоков в файле на порядок меньше, чем в стандартном алгоритме RC6, размер блока которого равен 16 байт. Например, в файле размером 12 572 162 байт, блоков по 16 байт + нули в конце файла – 785 761, в то время как блоков по 24 байт + нули в конце этого же файла входит только 523 841.

2. Также были зашифрованы/расшифрованы файлы без добавления расширения. После расшифрования подобного файла к нему только

требовалось добавить то расширение, которое было у исходного незашифрованного файла и запустить его соответствующим приложением.

3. Исходя из данных в таблице, можно сделать вывод, что алгоритм шифрования RC6 имеет высокую скорость шифрования/расшифрования файлов и соответствует ~0,6 Мб/сек в стандартном варианте и ~0,9 Мб/сек в модифицированном. Однако при увеличении размера файла скорость зашифрования/расшифрования постепенно снижается.

4. При попытке удаления/замены хотя бы одного символа в тексте зашифрованного файла, то правильно и корректно расшифровать его с помощью того же ключа не представляется возможным.

5. Также был рассмотрен вариант шифрования при частичной замене содержимого исходного файла с использованием того же ключа: в результате замены нескольких символов в начале исходного файла было установлено, что зашифрованные файлы резко отличаются друг от друга. Это является подтверждением высокой степени перемешивания бит в результате работы алгоритма шифрования.

6. В результате тестирования были рассмотрены варианты использования ключа с буквенной и цифровой последовательностью. Выяснилось, что степень перемешивания бит при кодировании настолько высока, что не имеет значения какая любая комбинация цифр или букв используется при шифровании.

3.2.2 Подбор пароля в файлах

Время подбора пароля в зашифрованном файле не влияет на его размер и расширение, т.к. для сравнения берутся только два первых блока в файле. Значение имеет лишь длина ключа при зашифровании и его сложность.

В табл. 3 указаны используемые ключи при подборе пароля как цифровые, буквенные, их сочетание, а также некоторые дополнительные символы. Время при подборе указано в сек, мин и часах. Пароль искался без задержек и прорисовки текущего времени.

Табл. 3. Время подбора пароля с разными ключами

Ключи	**Методы поиска**	**FileCoder Standart**	**FileCoder Modified**
12345	По цифрам	<1с	<1с
999999	По цифрам	10с	33с
135792468	По цифрам	24м 30с	1ч 16м 22с
ZZZ	По буквам	<1с	<1с
WORD	По буквам	4с	13с
MOUSE	По буквам	1м 15с	3м 22с
MOUSE	По буквам и цифрам	3м 46с	12м 45с
A567B	По буквам и цифрам	32с	1м 43с
CHIPSET	По буквам	1ч 11м 42с	8ч 57м 10с
SERVER	По буквам	47м 08с	2ч 03м 42с

Результаты тестирования:

1. Как видно из результатов тестирования, подбор пароля при стандартном варианте RC6 ведется в несколько раз быстрее, чем при модифицированном. Это связано, прежде всего, с количеством регистров и процедурой генерации ключа. В модифицированном варианте алгоритма RC6, как известно, регистров 6 и усложнена и дополнена генерация ключа.

2. Если длина ключа кратна 4 байтам (8, 12, 16 байт и т.д.), то последний символ не влияет на результат расшифрования, он может быть абсолютно любым. Важно лишь присутствие этого байта в ключе, как следствие особенности алгоритма RC6.

3. Подбор пароля по цифирной последовательности ведется на много быстрее, чем по буквенной, т.к. цифр всего 10, а букв 32. При выборе метода «поиска по всему» подбор ведется по всем 256-и ASCII символам, включая табуляцию, пробел, пустой символ, перевод каретки и другие.

Итак, в данной работе была спроектирована и реализована программа шифрования/расшифрования данных по алгоритму RC6.

FileCoder обеспечивает надежную защиту данных от несанкционированного доступа. Используемый в программе криптографический алгоритм является устойчивым к любым возможным типам взлома: несмотря на многолетние усилия исследователей, эффективных атак на него не обнаружено. Программа может использоваться как организациями для защиты коммерческой тайны, так и частными лицами для хранения в тайне личной информации, а также и в учебных целях.

Результаты тестирования программы показали ее правильную и устойчивую работу. Полученные результаты работы программы были проанализированы и позволяют сделать вывод о том, что поставленная задача была успешна решена.

Реализация системы проводилась с использованием инструментальных средств Borland Delphi 7.0. Размер приложения равен около 850 кб. Программа не требует инсталляции и не вносит изменения в реестр. Технические требования: персональный компьютер с операционной системой Windows 9x/ME/2000/XP. При написании программы основное внимание было уделено удобству работы пользователя и построению дружественного интерфейса.

ЗАКЛЮЧЕНИЕ

В настоящее время, в эру развития компьютерной и вычислительной техники, когда любой документ или файл хранится на электронных носителях, который может проявлять интерес у лиц, явно заинтересованных в его содержании и в дальнейшем распространении. Ни один файл на локальном компьютере, в сети и даже в Интернете, не защищен от посягательств как со стороны других пользователей, так и явных нарушителей. Поэтому требуются, как обязательный фактор во многих организациях как коммерческих, так и государственных, организационные меры по защите и сохранению в первозданном виде любых данных, которые могут быть подвержены изучению как со стороны любопытных пользователей, так нападению профессиональных хакеров.

Для решения задач обеспечения безопасности в информационных технологиях необходимо:

- защитить информацию при ее сохранении, обработке и передаче по сети;
- подтвердить подлинность объектов данных и пользователей (аутентификация сторон, устанавливающих связь);
- обнаружить и предупредить нарушение целостности объектов данных;
- защитить технические устройства и помещения;
- защитить конфиденциальную информацию от утечки и от внедренных электронных устройств съема информации;
- защитить программные продукты от внедрения программных закладок и вирусов;
- защитить от несанкционированного доступа к информационным ресурсам и техническим средствам сети, в том числе и к средствам управления, чтобы предотвратить снижение уровня защищенности информации и самой сети в целом;

- организовать требующиеся мероприятия, направленные на обеспечение сохранности конфиденциальных данных.

Конкретная реализация общих принципов обеспечения информационной безопасности может выражаться в организационных либо технических мерах защиты информации.

Одна из таких мер – криптографическая защита, использующая надежные и проверенные алгоритмы шифрования данных. Используя криптографическое шифрование любые данные зашифровываются до нечитабельного текста и невозможности открытия соответствующим приложением. В любом случае выбранный комплекс криптографических методов должен сочетать как удобство, гибкость и оперативность использования, так и надежную защиту от злоумышленников.

Применение методов криптографической защиты характерно для решения подавляющего большинства проблем безопасности. Аутентификация, шифрование данных, контроль целостности, электронная цифровая подпись – понятия, хорошо знакомые сегодня достаточно широкому кругу разработчиков и пользователей. Однако большинство пользователей пренебрегают элементарной защитой как своих личных данных, которые в большинстве случае не представляют никакого интереса, так и компьютера в целом. Многие пользователи даже не слышали или не задумывались о первостепенной защите информации на компьютере как установка антивируса и файерволла, не говоря уже о принудительном шифровании данных одним из криптостойких алгоритмов. Такая халатность ведет рано или поздно к заражению данных на компьютере различными вирусами, а также замедление или выхода из строя самой операционной системы.

Применение криптографических методов защиты обеспечивает решение основных задач информационной безопасности. Этого можно добиться путем реализации следующих криптографических методов защиты как пользовательской и служебной информации, так и информационных ресурсов в целом:

- шифрование всего информационного трафика, передающегося через открытые сети передачи данных, и отдельных сообщений;
- криптографическая аутентификация устанавливающих связь разноуровневых объектов (имеются в виду уровни модели взаимодействия открытых систем);
- защита несущего данные трафика средствами имитозащиты (защиты от навязывания ложных сообщений) и электронно-цифровой подписи с целью обеспечения целостности и достоверности передаваемой информации;
- шифрование данных, представленных в виде файлов либо хранящихся в базе данных;
- применение затемняющей цифровой подписи для обеспечения неотслеживаемости действий клиента в платежных системах, основанных на понятии электронных денег.

После распространения компьютеров в деловой сфере практическая криптография сделала в своем развитии огромный скачок, причем сразу по нескольким направлениям:

- во-первых, были разработаны стойкие блочные алгоритмы с секретным ключом, предназначенные для решения классической задачи – обеспечения секретности и целостности, передаваемых или хранимых данных;
- во-вторых, были созданы методы решения новых, нетрадиционных задач сферы защиты информации, наиболее известными из которых являются задача подписи цифрового документа и открытого распределения ключей.

От того, насколько методически правильно будет организована криптографическая работа, зависит результат, цена которого – надежность и целостность сохранения данных.

БИБЛИОГРАФИЯ

1. Аграновский А.В, Хади Р.А. Практическая криптография: алгоритмы и их программирование. – М: «Солон-Р», 2002.

2. Аграновский А.В., Балакин А.В., Хади Р.А. Классические шифры и методы их криптоанализа. – М: Информационные технологии, 2001.

3. Баричев С.Г. Криптография без секретов. – М: «Горячая линия - Телеком», 2000.

4. Баричев С.Г., Серов Р.Е. Основы современной криптографии. – М: «Горячая линия - Телеком», 2001.

5. Беляев А.В. Методы и средства защиты информации (курс лекций). – СПб: ЧФ СПбГТУ, 2000.

6. Варфоломеев А.А, Жуков А.Е., Пудовкина М.А. Поточные криптосистемы. Основные свойства и методы анализа стойкости. – М: ПАИМС, 2000.

7. Введение в криптографию. Под общ. ред. Ященко В.В. – М: МЦНМО-ЧеРо, 1998.

8. Винокуров А. Алгоритм шифрования ГОСТ 28147-89, его использование и реализация для компьютеров платформы Intel x86. Статья.

9. Герасименко В.А. Защита информации в автоматизированных системах обработки данных. – М: Энергоатомиздат, 2000.

10. Жельников В. Криптография от папируса до компьютера. – М: ABF, 1996.

11. Казарин О.В. Безопасность программного обеспечения компьютерных систем. – М: МГУЛ, 2003.

12. Липаев В.В. Программно-технологическая безопасность информационных систем. – М: Русская редакция, 2001.

13. Лукацкий А.В. Как обеспечить подлинность электронных документов. Статья.

14. Мафтик С. Механизмы защиты в сетях ЭВМ. – М: Мир, 1995.

15. Медведовский И.Д., Семьянов П.В., Платонов В.В. Атака через Интернет. – СПб: 1999.

16. Мельников В.В. Защита информации в компьютерных системах. – М: Финансы и статистика, 1997.

17. Молдовян А.А., Молдовян Н.А., Советов Б.Я. Криптография. – СПб: «Лань», 2000.

18. Молдовян Н.А. Скоростные блочные шифры. – Спб: Издательство СПбГТУ, 1998.

19. Нечаев В.И. Элементы криптографии (Основы теории защиты информации). – М: Высшая школа, 1999.

20. Основы криптозащиты АСУ. Под. Ред. Б.П. Козлова. – М: МО, 1996.

21. Петров А.А. Компьютерная безопасность. Криптографические методы защиты. – М: ДМК, 2000.

22. Программно-аппаратные средства обеспечения информационной безопасности. Защита программ и данных: Учебное пособие для вузов / П.Ю. Белкин. – М: «Радио и связь», 1999.

23. Романцев Ю.В., Тимофеев П.А. Защита информации в компьютерных системах и сетях. – М: «Радио и связь», 2001.

24. Саломаа А. Криптография с открытым ключом. Пер. с англ. – М: Мир, 1995.

25. Столлингс В. Криптография и защита сетей. – М: Вильямс, 2001.

26. Устинов Г.Н. Основы информационной безопасности. – М: «Синтег», 2000.

27. Ухлинов А.М. Управление безопасностью информации в автоматизированных системах. – М: МИФИ, 1996.

28. Чандлер Дж. “Cryptography 101”. Статья.

29. Чмора А. Современная прикладная криптография. – М: «Гелиос АРБ», 2001.

30. Щербаков Л.Ю., Доманшев А.В. Прикладная криптография. Использование и синтез криптографических интерфейсов. – М: Русская редакция, 2003.

ПРИЛОЖЕНИЕ. ОПИСАНИЕ РАБОТЫ ШИФРУЮЩЕЙ ПРОГРАММЫ FILECODER

Работа с программой

Программа FileCoder предназначена для шифрования и расшифрования файлов. В качестве шифрования используется модифицированный алгоритм RC6. Это блочный алгоритм шифрования с симметричным ключом т.е. один ключ используется как для шифрования, так и для расшифрования данных. Длина ключа (пароля) может составлять от 1 до 255 символов.

FileCoder обеспечивает надежную защиту данных от несанкционированного доступа. Используемый в программе криптографический алгоритм является устойчивым к любым возможным типам взлома: несмотря на многолетние усилия исследователей, эффективных атак на него не обнаружено. Программа может использоваться как организациями для защиты коммерческой тайны, так и частными лицами для хранения в тайне личной информации (Рис. 1).

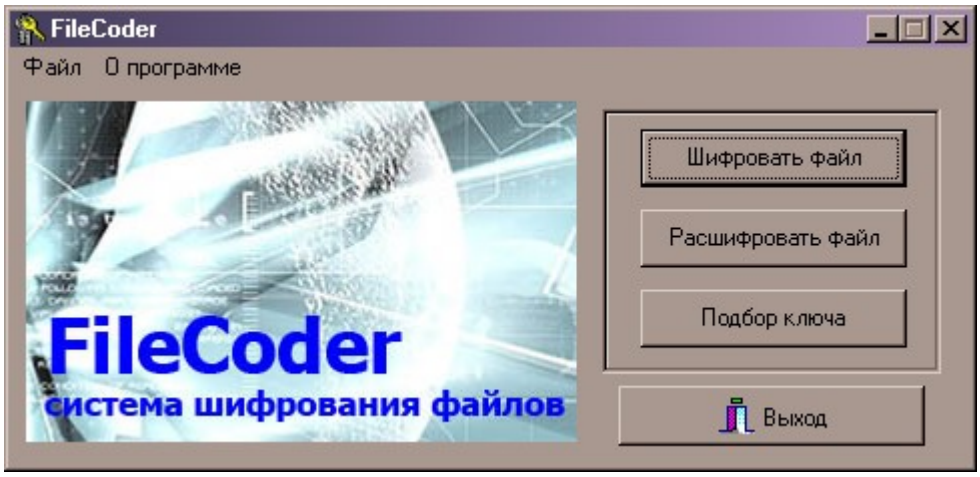


Рис. 1. Главное окно программы FileCoder

Шифрование файлов

Для того, чтобы зашифровать файл, нажмите кнопку «Шифровать файл» в окне программы или воспользуйтесь пунктом меню «Файл»->«Шифровать». В открывшемся окне «Шифрование файла» выберите нужный исходный файл,

который будет подвержен шифрованию. В качестве конечного файла выберите любой файл, неважно, существует ли он на диске или нет. В конечном файле можно не указывать расширение. В блоке «Количество раз шифрования файла» укажите сколько раз следует зашифровать конечный файл, набрав необходимое число в строку ввода, воспользовавшись стрелками прокрутки или выбрав готовое число кнопкой выбора. Введите ключ для шифрования файла. Рекомендуется использовать ключ длиной 8 и более символов (при увеличении длины ключа файл будет более устойчив к взлому). Ключом может быть любая как буквенно-цифровая последовательность символов латинского и русского алфавита, так и другие символы из ASCII таблицы. Для расшифрования применяется тот же самый ключ, который был использован для шифрования (Рис. 2).

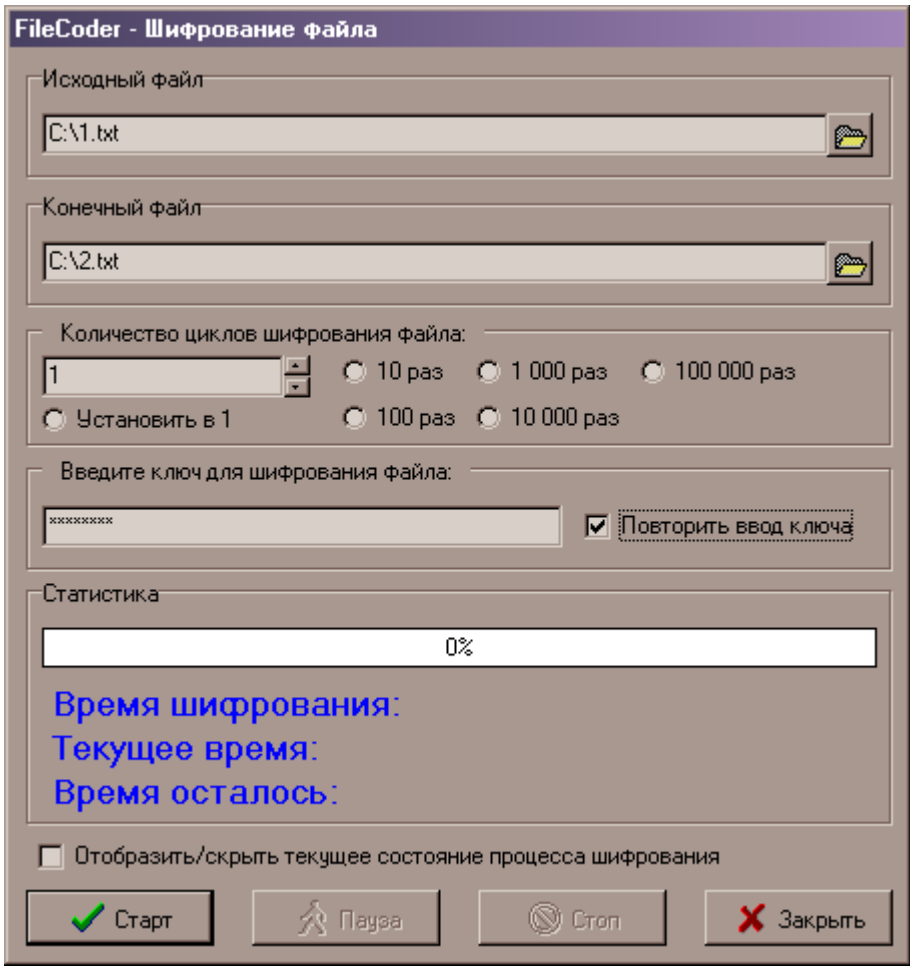


Рис. 2. Окно шифрования файла

Если поставить галочку «Повторить ввод ключа», то потребуется подтверждение при вводе ключа. Подтверждение может быть необходимо для исключения возможных опечаток (особенно если ключ слишком длинный и использует случайную последовательность символов). Если ключ и его подтверждение не совпадают, программа сообщит об этом и надо будет ввести ключ заново (Рис. 3).

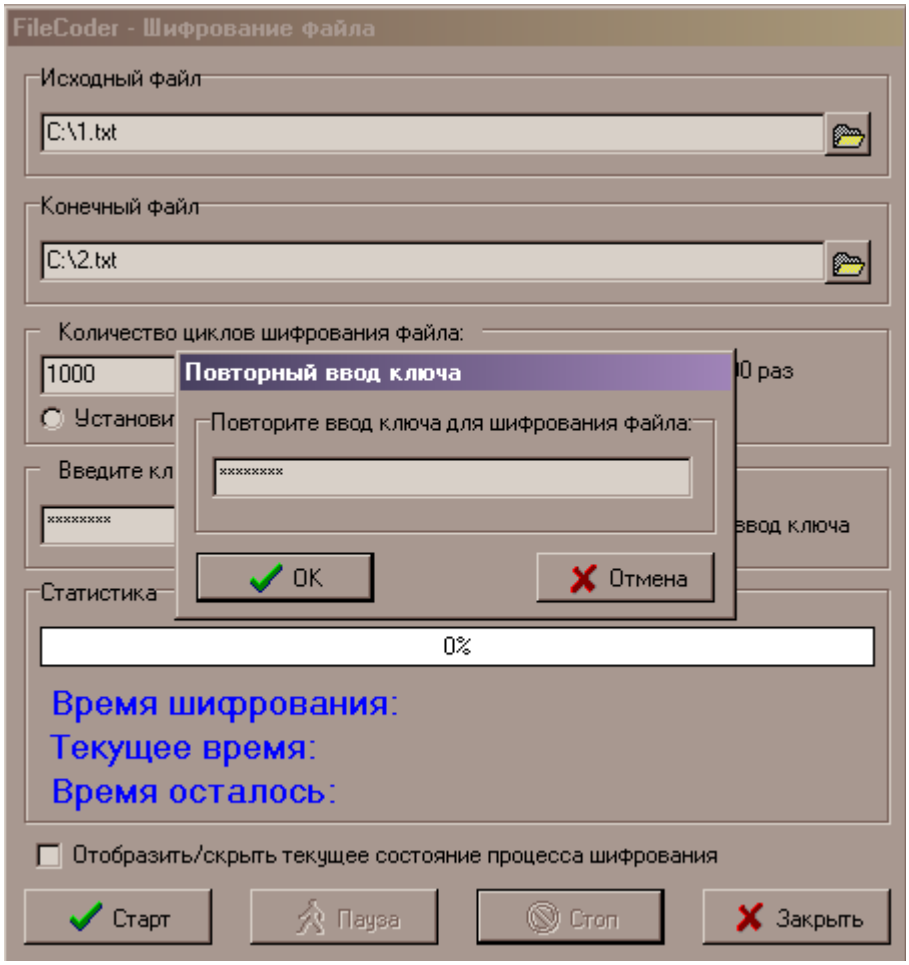


Рис. 3. Окно повторного ввода ключа

После успешного ввода ключа начинается процесс шифрования. Ход процесса отображается в статистике с помощью индикации состояния процесса и отсчетом времени, показывающим сколько необходимо времени для шифрования файла, текущее состояние времени и сколько осталось времени до конца шифрования файла (Рис. 4).

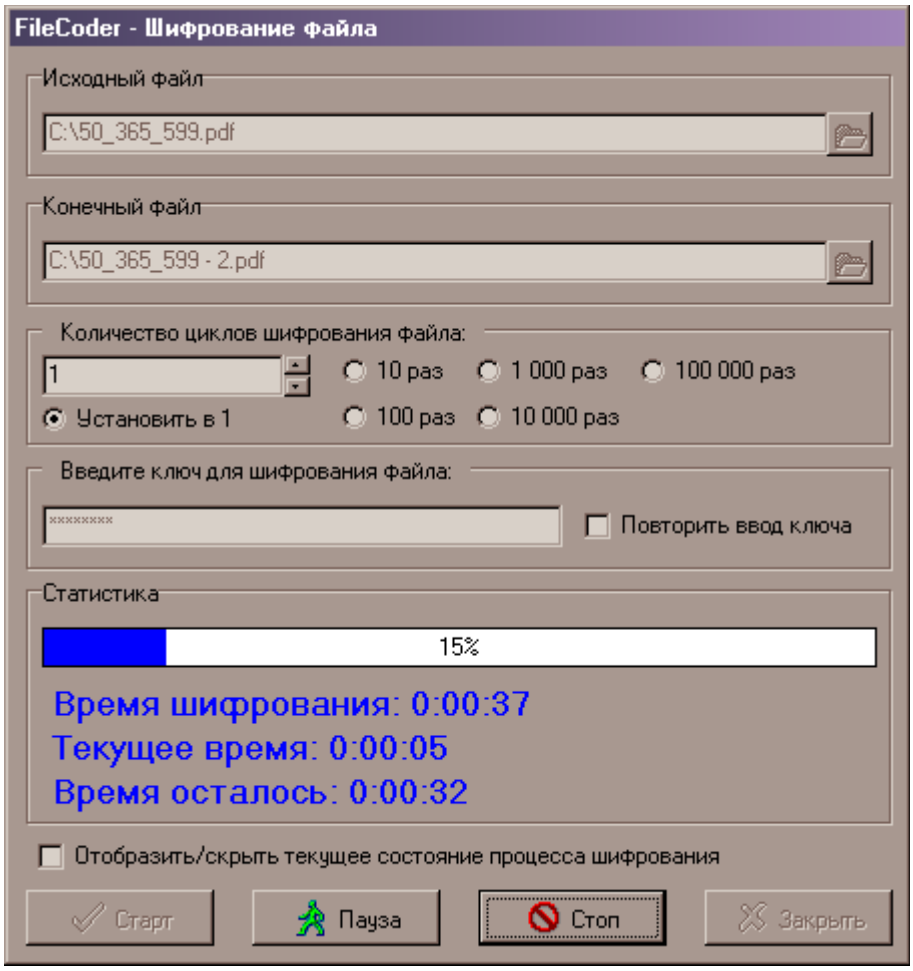


Рис. 4. Процесс шифрования файла

Становятся доступные кнопки «Пауза» и «Стоп», которые позволяют приостановить или отменить процесс шифрования файла соответственно.

При установке галочки «Отобразить/скрыть текущее состояние процесса шифрования» появляется группировка объектов текущего состояния процесса шифрования с указанием шестнадцатеричных значений данных на входе и выходе по всем 6-ти блоках (Рис. 5).

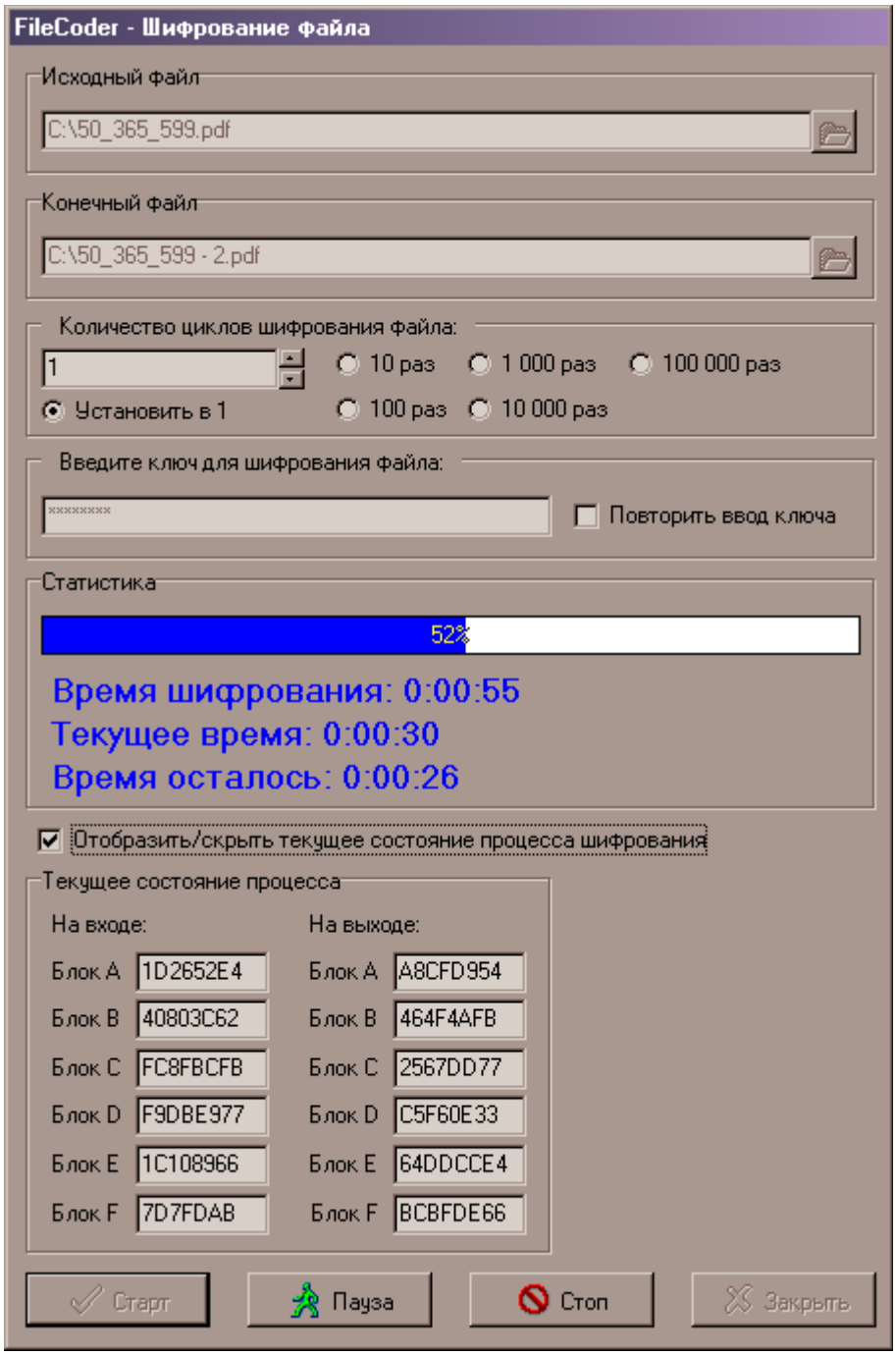


Рис. 5. Текущее состояние процесса шифрования

По завершении процесса шифрования программа выдаст соответствующее сообщение и информирует сколько раз был зашифрован файл (Рис. 6).

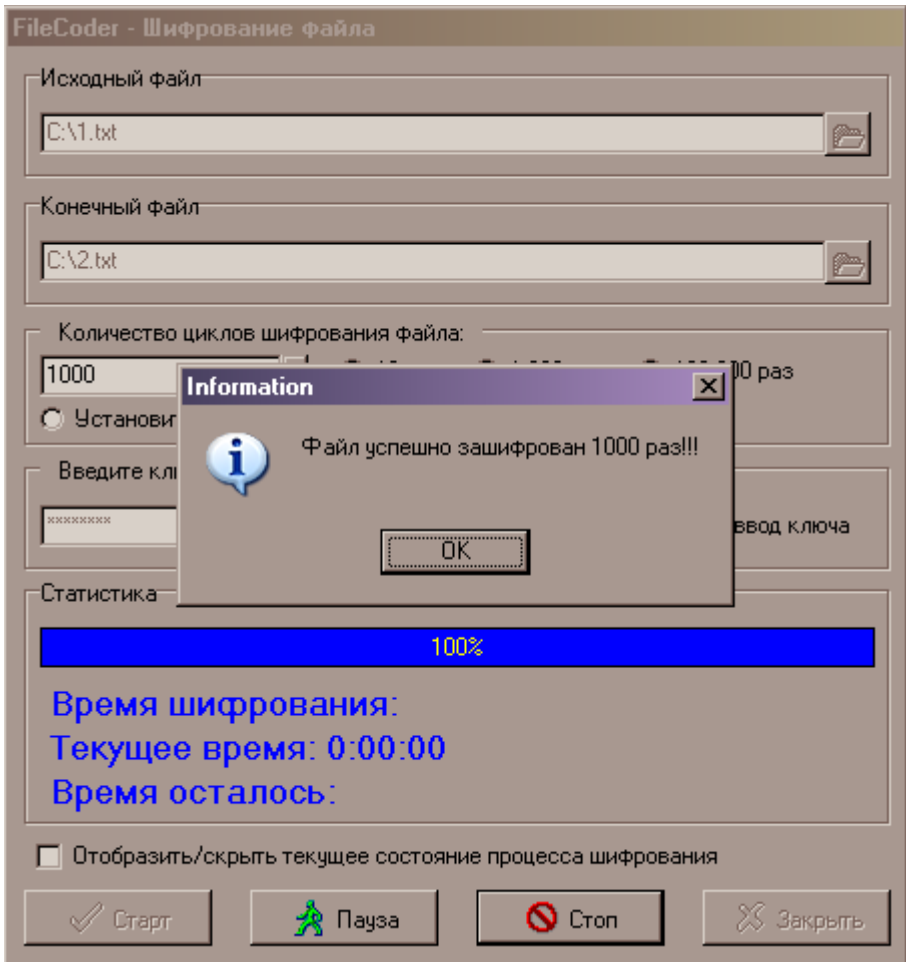


Рис. 6. Окончание шифрования файла

Расшифрование файлов

Для расшифрования файла нажмите кнопку «Расшифровать файлы» в окне программы или воспользуйтесь пунктом меню «Файл»->«Расшифровать». В открывшемся окне «Расшифрование файла» выберите нужный исходный файл, который был ранее зашифрован. В качестве конечного файла выберите любой файл, неважно, существует ли он на диске или нет. В конечном файле можно не указывать расширение. Затем введите ключ для расшифрования. Если же для расшифрования файла был введен неверный ключ, то программа сообщит об этом (Рис. 7).

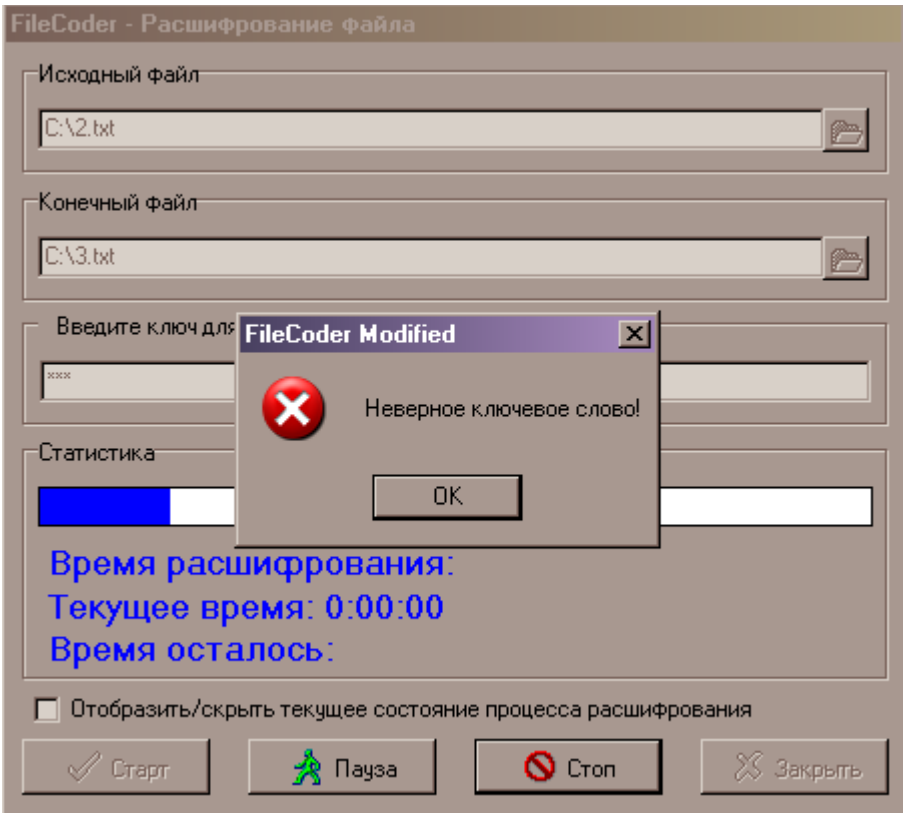


Рис. 7. Ошибка при вводе ключа при расшифровании файла

После успешного ввода ключа начинается процесс расшифрования. Ход процесса отображается в статистике с помощью индикации состояния процесса и отсчетом времени, показывающим сколько необходимо времени для расшифрования файла, текущее состояние времени и сколько осталось времени до конца расшифрования файла.

Становятся доступные кнопки «Пауза» и «Стоп», которые позволяют приостановить или отменить процесс расшифрования файла соответственно.

При установке галочки «Отобразить/скрыть текущее состояние процесса расшифрования» появляется группировка объектов текущего состояния процесса расшифрования с указанием шестнадцатеричных значений данных на входе и выходе по всем 6-ти блоках (Рис. 8).

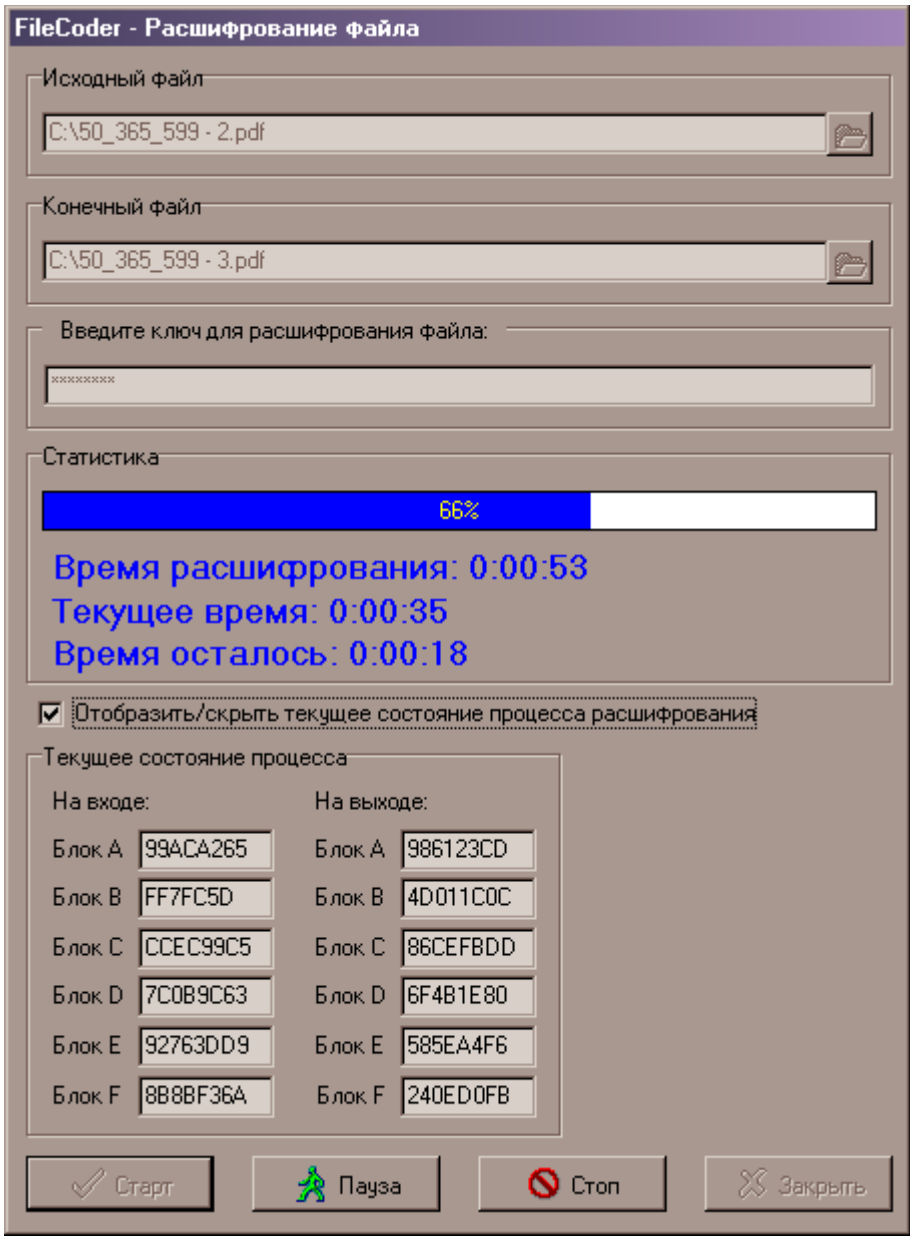


Рис. 8. Текущее состояние процесса расшифрования

По завершении процесса расшифрования программа выдаст соответствующее сообщение и информирует сколько раз был расшифрован файл (Рис. 9).

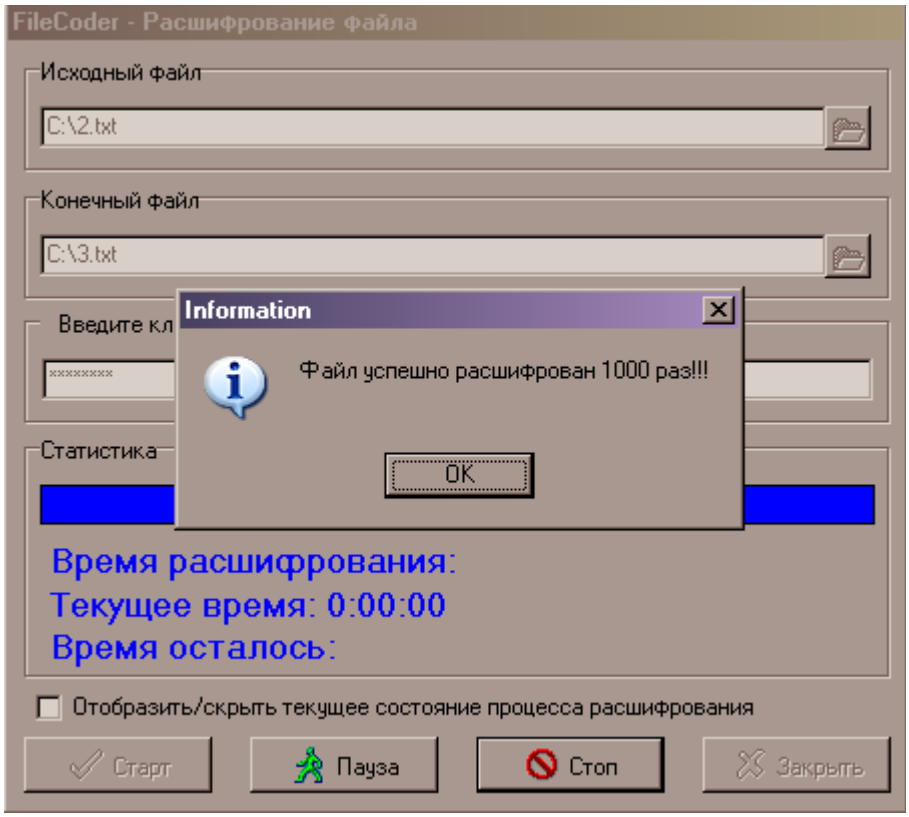


Рис. 9. Окончание расшифрования файла

Подбор ключа

Подбора ключа используется для ранее зашифрованных по данному алгоритму файлов с использованием программы FileCoder в случае утери необходимого ключа для расшифрования. Не имеет значения какого размера и сколько раз шифровался исходный незашифрованный файл. Размер файла и количество раз шифрования не влияют на процесс подбора ключа.

Для проведения подбора ключа нажмите кнопку «Подбор ключа» в окне программы или воспользуйтесь пунктом меню «Файл»->«Подбор ключа». В открывшемся окне «Подбор ключа» выберите нужный исходный файл, который ранее был зашифрован. В блоке «Методы поиска» необходимо выбрать один из нескольких методов поиска по заданному критерию. Критерий «По списку» использует при подборе ключа символы, указанные рядом в строке ввода (Рис. 10).

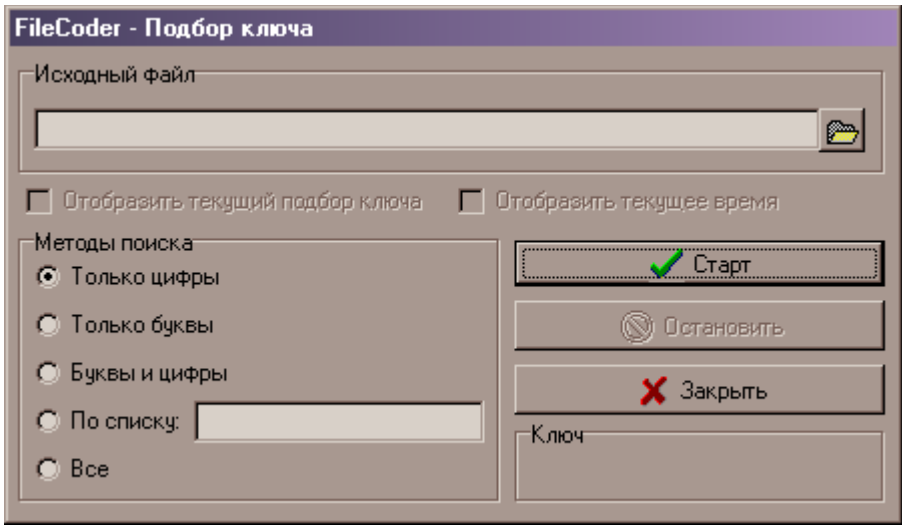


Рис. 10. Окно подбора ключа

При указании необходимого метода поиска запускается процесс подбора ключа. Представляется возможным поставить одну из двух галочек. Первая «Отобразить текущий подбор ключа» показывает текущий на данный промежуток времени подбор ключа. Данная функция очень удобна, т.к. позволяет посмотреть на каком этапе находится подбор ключа и позволяет с ориентироваться во времени ожидания (Рис. 11).

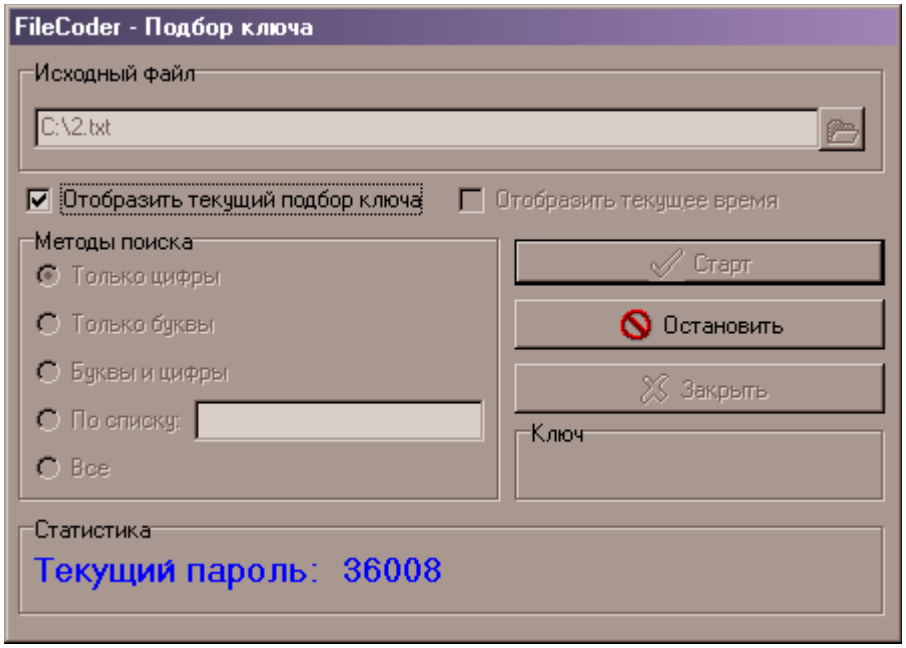


Рис. 11. Процесс отображения текущего ключа

При выборе второй галочки «Отобразить текущее время», показывает сколько уже времени ведется подбор ключа. При снятии обоих галочек «прячется» статистика, тем самым освобождая ресурсы процессора и заметно уменьшая время на подбор ключа (Рис. 12).

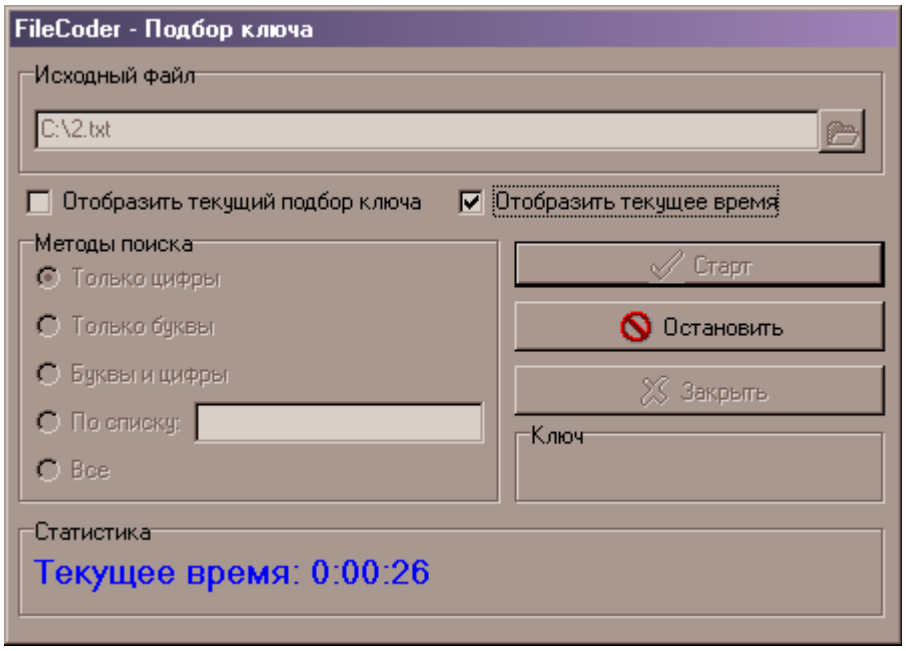


Рис. 12. Процесс отображения текущего времени

Если пароль был успешно найден, то программа отобразит его и сколько времени прошло с начала момента запуска подбора пароля (Рис. 13).

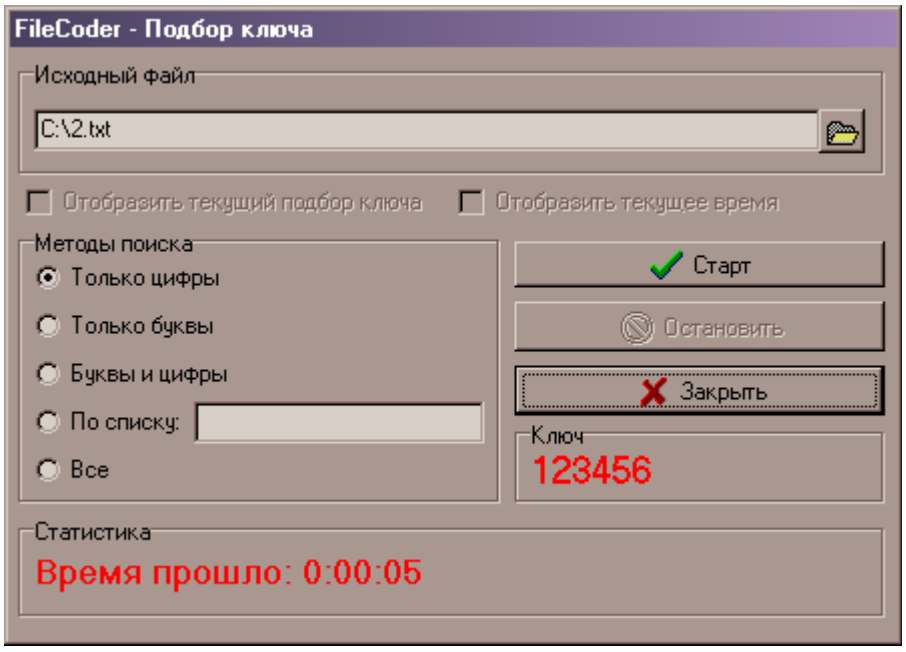


Рис. 13. Окончание процесса подбора ключа