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

Андрей Винокуров (avin@chat.ru).

Предлагаемая вашему вниманию статья содержит описание алгоритма, принятого в качестве стандарта шифрования в Российской Федерации, и его реализации для процессоров семейства Intel x86, а также обсуждение различных вопросов его практического использования. Данная статья содержит существенно переработанные и дополненные материалы, опубликованные в журнале «Монитор» №1,5 в 1995 году.

С о д е р ж а н и е

Вместо предисловия.

То, что информация имеет ценность, люди осознали очень давно, – недаром переписка сильных мира сего издавна была объектом пристального внимания их недругов и друзей. Тогда-то и возникла задача защиты этой переписки от чрезмерно любопытных глаз. Древние пытались использовать для решения этой задачи самые разнообразные методы, и одним из них была тайнопись – умение составлять сообщения таким образом, чтобы его смысл был недоступен никому кроме посвященных в тайну. Есть свидетельства тому, что искусство тайнописи зародилось еще в доантичные времена. На протяжении всей своей многовековой истории, вплоть до совсем недавнего времени, это искусство служило немногим, в основном верхушке общества, не выходя за пределы резиденций глав государств, посольств и – конечно же! – разведывательных миссий. И лишь несколько десятилетий назад все изменилось коренным образом – информация приобрела самостоятельную коммерческую ценность и стала широко распространенным, практически рядовым товаром. Ее производят, хранят, транспортируют, продают и покупают, а значит – воруют и подделывают – и, следовательно, ее необходимо защищать. Современное общество все в большей степени становится информационно–обусловленным, успех любого вида деятельности все сильней зависит от обладания определенными сведениями и от отсутствия их у конкурентов. И чем сильней проявляется указанный эффект, тем больше потенциальные убытки от злоупотреблений в информационной сфере, и тем больше потребность в защите информации. Одним словом, возникновение индустрии обработки информации с железной необходимостью привело к возникновению индустрии средств защиты информации.

Среди всего спектра методов защиты данных от нежелательного доступа особое место занимают криптографические методы. В отличие от других методов, они опираются лишь на свойства самой информации и не используют свойства ее материальных носителей, особенности узлов ее обработки, передачи и хранения. Образно говоря, криптографические методы строят барьер между защищаемой информацией и реальным или потенциальным злоумышленником из самой информации. Конечно, под криптографической защитой в первую очередь, – так уж сложилось исторически, – подразумевается шифрование данных. Раньше, когда эта операция выполнялось человеком вручную или с использованием различных приспособлений, и при посольствах содержались многолюдные отделы шифровальщиков, развитие криптографии сдерживалось проблемой реализации шифров, – ведь придумать можно было все что угодно, но как это реализовать… . Появление цифровых электронно-вычислительных машин, приведшее в конечном итоге к созданию мощной информационной индустрии, изменило все коренным образом и в этой сфере. С одной стороны, взломщики шифров получили в свои руки чрезвычайно мощное орудие, с другой стороны, барьер сложности реализации исчез, и для создателей шифров открылись практически безграничные перспективы. Все это определило стремительный прогресс криптографии в последние десятилетия.

Как всякое уважающее себя государство, Российская Федерация имеет свой стандарт шифрования. Этот стандарт закреплен ГОСТом №28147-89, принятом, как явствует из его обозначения, еще в 1989 году в СССР. Однако, без сомнения, история этого шифра гораздо более давняя. Стандарт родился предположительно в недрах восьмого главного управления КГБ СССР, преобразованного ныне в ФАПСИ. Мне довелось беседовать с людьми, утверждавшими, что еще в 70-х годах они участвовали в проектах создания программных и аппаратных реализаций этого шифра для различных компьютерных платформ. В те времена он имел гриф «сов. секретно», позже гриф был изменен на «секретно», затем снят совсем. На моем экземпляре ГОСТа стояла лишь скромная пометка «ДСП». К сожалению, в отличие от

самого стандарта, история его создания и критерии проектирования шифра до сих пор остаются тайной за семью печатями.

Возможное использование ГОСТа в Ваших собственных разработках ставит ряд вопросов. Вопрос первый – нет ли юридических препятствий для этого. Ответ здесь простой – таких препятствий нет и вы можете свободно использовать ГОСТ, он не запатентован, следовательно, не у кого спрашивать разрешения. Более того, вы имеете на это полное моральное право, – либо вы сами оплатили эту разработку своими налогами, явными и скрытыми, либо, если вы принадлежите к более молодому поколению, это сделали ваши родители. Известный указ Президента России №334 от 03.04.95 и соответствующие постановления правительства ничего нового не вносят в эту картину. Хотя они формально и запрещают разработку систем, содержащих средства криптозащиты юридическими и физическими лицами, не имеющими лицензии на этот вид деятельности, но реально указ распространяется лишь на случай государственных секретов, данных, составляющих банковскую тайну и т.п., словом, он действует только там, где нужна бумажка, что «данные защищены».

Хорошо, с правомочностью применения ГОСТа разобрались, теперь остановимся на вопросе целесообразности – прежде всего, можем ли мы доверять этому порождению мрачной Лубянки, не встроили ли товарищи чекисты лазеек в алгоритмы шифрования? Это весьма маловероятно, так как ГОСТ создавался в те времена, когда было немыслимо его использование за пределами государственных режимных объектов. С другой стороны, стойкость криптографического алгоритма нельзя подтвердить, ее можно только опровергнуть взломом. Поэтому, чем старше алгоритм, тем больше шансов на то, что, если уж он не взломан до сих пор, он не будет взломан и в ближайшем обозримом будущем. В этом свете все разговоры о последних «оригинальных разработках талантливых ребят» в принципе не могут быть серьезными – каждый шифр должен выдержать проверку временем. Но ведь шифров, выдержавших подобную проверку, заведомо больше одного – кроме ГОСТа ведь есть еще и DES, его старший американский братец, есть и другие шифры. Почему тогда ГОСТ? Конечно, во многом это дело личных пристрастий, но надо помнить еще и о том, что ГОСТ по большинству параметров превосходит все эти алгоритмы, в том числе и DES. И, в конце концов, где же наш Российский патриотизм?!

Широкому использованию ГОСТа в разработках Российских программистов мешает, по моему мнению, недостаток опубликованной информации о нем, а также некий ореол таинственности, сложившийся вокруг него и искусно кем-то поддерживаемый. На самом деле ничего сложного в шифре нет, он доступен для понимания и реализации программисту любого уровня, но, как и во всем прочем, для создания действительно ***хорошей реализации*** надо быть профессионалом. Я работал с ГОСТом как программист четыре года, – с 91 по 94 год, – и за это время у меня получилась весьма удачная его программная реализация для процессоров семейства Intel x86, приближающаяся по быстродействию к возможному для младших процессоров оптимуму.

Целью настоящей статьи является знакомство всех заинтересованных с самим алгоритмом и его реализацией на платформе Intel x86. Разработанную мной реализацию ГОСТа я предоставляю в ***общественную собственность****, ее может использовать каждый без каких-либо ограничений. Текст настоящей статьи может неограниченно распространяться в печатном и электронном виде в том и только в том случае, если это не сопряжено прямо или косвенно с извлечением прибыли, в противном случае требуется мое письменное разрешение.*

Статья состоит из четырех частей. Первая часть содержит описание, а вторая – обсуждение алгоритма, третья и четвертая части содержат соответственно описание его реализации и обсуждение некоторых аспектов его применения. Итак, начнем…

1. Описание алгоритма.

1.1. Термины и обозначения.

Описание стандарта шифрования Российской Федерации содержится в очень интересном документе, озаглавленном «Алгоритм криптографического преобразования данных ГОСТ 28147-89». То, что в его названии вместо термина «шифрование» фигурирует более общее понятие «***криптографическое преобразование***», вовсе не случайно. Помимо нескольких тесно связанных между собой процедур шифрования, в документе описан один построенный на общих принципах с ними алгоритм выработки ***имитовставки***. Последняя является не чем иным, как криптографической контрольной комбинацией, то есть кодом, вырабатываемым из исходных данных с использованием секретного ключа с целью ***имитозащиты***, или защиты данных от внесения в них несанкционированных изменений.

На различных шагах алгоритмов ГОСТа данные, которыми они оперируют, интерпретируются и используются различным образом. В некоторых случаях элементы данных обрабатываются как массивы независимых битов, в других случаях – как целое число без знака, в третьих – как имеющий структуру сложный элемент, состоящий из нескольких более простых элементов. Поэтому во избежание путаницы следует договориться об используемых обозначениях.

Элементы данных в данной статье обозначаются заглавными латинскими буквами с наклонным начертанием (например, X). Через $|X|$ обозначается размер элемента данных X в битах. Таким образом, если интерпретировать элемент данных X как целое неотрицательное число, можно записать следующее неравенство: $0 \le X < 2^{|X|}$.

Если элемент данных состоит из нескольких элементов меньшего размера, то этот факт обозначается следующим образом: $X = (X_0, X_1, \ldots, X_{n-1}) = X_0 \,||\, X_1 \,||\, \ldots \,||\, X_{n-1}$. Процедура объединения нескольких элементов данных в один называется ***конкатенацией*** данных и обозначается символом «||». Естественно, для размеров элементов данных должно выполняться следующее соотношение: $|X| = |X_0| + |X_1| + \ldots + |X_{n-1}|$. При задании сложных элементов данных и операции конкатенации составляющие элементы данных перечисляются в порядке возрастания старшинства. Иными словами, если интерпретировать составной элемент и все входящие в него элементы данных как целые числа без знака, то можно записать следующее равенство:

$$(X_0, X_1, ..., X_{n-1}) = X_0 || X_1 || ... || X_{n-1} = X_0 + 2^{|X_0|}(X_1 + 2^{|X_1|}(...(X_{n-2} + 2^{|X_{n-2}|} X_{n-1})...)).$$

В алгоритме элемент данных может интерпретироваться как массив отдельных битов, в этом случае биты обозначаем той же самой буквой, что и массив, но в строчном варианте, как показано на следующем примере:

$$X = (x_0, x_1, \ldots, x_{n-1}) = x_0 + 2^1 \cdot x_1 + \ldots + 2^{n-1} \cdot x_{n-1}.$$

Если над элементами данных выполняется некоторая операция, имеющая логический смысл, то предполагается, что данная операция выполняется над соответствующими битами элементов. Иными словами $A \bullet B = (a_0 \bullet b_0, a_1 \bullet b_1, \ldots, a_{n-1} \bullet b_{n-1})$, где $n = |A| = |B|$, а символом «•» обозначается произвольная бинарная логическая операция; как правило, имеется ввиду операция ***исключающего или***, она же – операция суммирования по модулю 2: $a \oplus b = (a + b) \bmod 2$.

1.2. Логика построения шифра и структура ключевой информации ГОСТа.

Если внимательно изучить оригинал ГОСТ 28147–89, можно заметить, что в нем содержится описание алгоритмов нескольких уровней. На самом верхнем находятся практические алгоритмы, предназначенные для шифрования массивов данных и выработки для них имитовставки. Все они опираются на три алгоритма низшего уровня, называемые в тексте ГОСТа ***циклами***. Эти фундаментальные алгоритмы упоминаются в данной статье как ***базовые циклы***, чтобы отличать их от всех прочих циклов. Они имеют следующие названия и обозначения, последние приведены в скобках и смысл их будет объяснен позже:

- **цикл зашифрования (32-З);**
- **цикл расшифрования (32-Р);**
- **цикл выработки имитовставки (16-З).**

В свою очередь, каждый из базовых циклов представляет собой многократное повторение одной единственной процедуры, называемой для определенности далее в настоящей работе ***основным шагом криптопреобразования***.

Таким образом, чтобы разобраться в ГОСТе, надо понять три следующие вещи:
а) что такое ***основной шаг*** криптопреобразования;
б) как из ***основных шагов*** складываются ***базовые циклы***;
в) как из трех ***базовых циклов*** складываются все практические алгоритмы ГОСТа.

Прежде чем перейти к изучению этих вопросов, следует поговорить о ключевой информации, используемой алгоритмами ГОСТа. В соответствии с принципом Кирхгофа, которому удовлетворяют все современные известные широкой общественности шифры, именно ее секретность обеспечивает секретность зашифрованного сообщения. В ГОСТе ключевая информация состоит из двух структур данных. Помимо собственно ***ключа***, необходимого для всех шифров, она содержит еще и ***таблицу замен***. Ниже приведены основные характеристики ключевых структур ГОСТа.

1. ***Ключ*** является массивом из восьми 32-битовых элементов кода, далее в настоящей работе он обозначается символом $\boldsymbol{K}$: $\boldsymbol{K} = \{\boldsymbol{K_i}\}_{0 \le i \le 7}$. В ГОСТе элементы ключа используются как 32-разрядные целые числа без знака: $0 \le \boldsymbol{K_i} \le 2^{32}$. Таким образом, размер ключа составляет $32 \cdot 8 = 256$ бит или 32 байта.

2. ***Таблица замен*** может быть представлена в виде матрицы размера 8×16, содержащей 4-битовые элементы, которые можно представить в виде целых чисел от 0 до 15. Строки ***таблицы замен*** называются ***узлами замен***, они должны содержать различные значения, то есть каждый ***узел замен*** должен содержать 16 различных чисел от 0 до 15 в произвольном порядке. В настоящей статье таблица замен обозначается символом $\boldsymbol{H}$: $\boldsymbol{H} = \{\boldsymbol{H_{i,j}}\}_{0 \le j \le 15}^{0 \le i \le 7}$, $0 \le \boldsymbol{H_{i,j}} \le 15$. Таким образом, общий объем таблицы замен равен: 8 узлов × 16 элементов/узел × 4 бита/элемент = 512 бит или 64 байта.

1.3. Основной шаг криптопреобразования.

Основной шаг криптопреобразования по своей сути является оператором, определяющим преобразование 64-битового блока данных. Дополнительным параметром этого оператора является 32-битовый блок, в качестве которого используется какой-либо элемент ключа. Схема алгоритма основного шага приведена на рисунке 1. Ниже даны пояснения к алгоритму основного шага:

Шаг 0. Определяет исходные данные для основного шага криптопреобразования:

- $\boldsymbol{N}$ – преобразуемый 64-битовый блок данных, в ходе выполнения шага его младшая ($\boldsymbol{N}_1$) и старшая ($\boldsymbol{N}_2$) части обрабатываются как отдельные 32-битовые целые числа без знака. Таким образом, можно записать $\boldsymbol{N}=(\boldsymbol{N}_1,\boldsymbol{N}_2)$.

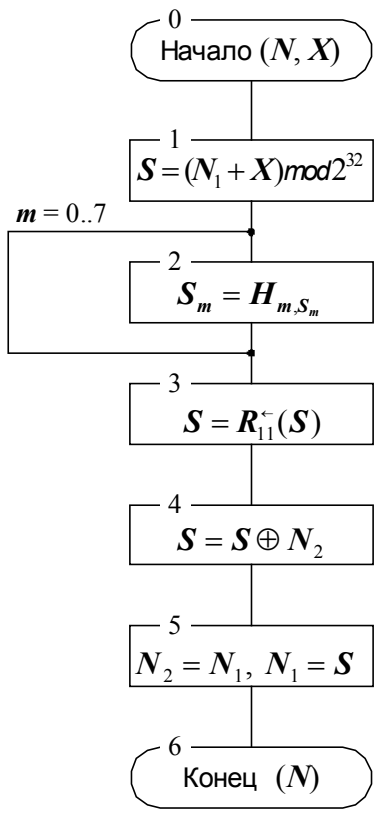

Рис. 1. Схема основного шага криптопреобразования алгоритма ГОСТ 28147-89.

- X – 32-битовый элемент ключа;

Шаг 1. Сложение с ключом. Младшая половина преобразуемого блока складывается по модулю 2^{32} с используемым на шаге элементом ключа, результат передается на следующий шаг;

Шаг 2. Поблочная замена. 32-битовое значение, полученное на предыдущем шаге, интерпретируется как массив из восьми 4-битовых блоков кода: $S = (S_0,S_1,S_2,S_3,S_4,S_5,S_6,S_7)$. Далее значение каждого из восьми блоков заменяется новым, которое выбирается по таблице замен следующим образом: значение блока S_i меняется на S_i-тый по порядку элемент (нумерация с нуля) i-того узла замен (т.е. i-той строки таблицы замен, нумерация также с нуля). Другими словами, в качестве замены для значения блока выбирается элемент из таблицы замен с номером строки, равным номеру заменяемого блока, и номером столбца, равным значению заменяемого блока как 4-битового целого неотрицательного числа. Теперь становится понятным размер таблицы замен: число строк в ней равно числу 4-битовых элементов в 32-битовом блоке данных, то есть восьми, а число столбцов равно числу различных значений 4-битового блока данных, равному как известно 2^4, шестнадцати.

Шаг 3. Циклический сдвиг на 11 бит влево. Результат предыдущего шага сдвигается циклически на 11 бит в сторону старших разрядов и передается на следующий шаг. На схеме алгоритма символом $R_{11}^{\leftarrow}$ обозначена функция циклического сдвига своего аргумента на 11 бит влево, т.е. в сторону старших разрядов.

Шаг 4. Побитовое сложение: значение, полученное на шаге 3, побитно складывается по модулю 2 со старшей половиной преобразуемого блока.

Шаг 5. Сдвиг по цепочке: младшая часть преобразуемого блока сдвигается на место старшей, а на ее место помещается результат выполнения предыдущего шага.

Шаг 6. Полученное значение преобразуемого блока возвращается как результат выполнения алгоритма основного шага криптопреобразования.

1.4. Базовые циклы криптографических преобразований.

Как отмечено в начале настоящей статьи, ГОСТ относится к классу блочных шифров, то есть единицей обработки информации в нем является блок данных. Следовательно, вполне логично ожидать, что в нем будут определены алгоритмы для криптографических преобразований, то есть для зашифрования, расшифрования и «учета» в контрольной комбинации одного блока данных. Именно эти алгоритмы и называются ***базовыми циклами*** ГОСТа, что подчеркивает их фундаментальное значение для построения этого шифра.

Базовые циклы построены из ***основных шагов*** криптографического преобразования, рассмотренного в предыдущем разделе. В процессе выполнения основного шага используется только один элемент ключа, в то время как ключ ГОСТ содержит восемь таких элементов. Следовательно, чтобы ключ был использован полностью, каждый из базовых циклов должен многократно выполнять основной шаг с различными его элементами. Вместе

с тем кажется вполне естественным, что в каждом базовом цикле все элементы ключа должны быть использованы одинаковое число раз, по соображениям стойкости шифра это число должно быть больше одного.

Все сделанные выше предположения, опирающиеся просто на здравый смысл, оказались верными. Базовые циклы заключаются в многократном выполнении *основного шага* с использованием разных элементов ключа и отличаются друг от друга только числом повторения шага и порядком использования ключевых элементов. Ниже приведен этот порядок для различных циклов.

1. Цикл зашифрования 32-З:

$\boldsymbol{K}_0,\boldsymbol{K}_1,\boldsymbol{K}_2,\boldsymbol{K}_3,\boldsymbol{K}_4,\boldsymbol{K}_5,\boldsymbol{K}_6,\boldsymbol{K}_7,\boldsymbol{K}_0,\boldsymbol{K}_1,\boldsymbol{K}_2,\boldsymbol{K}_3,\boldsymbol{K}_4,\boldsymbol{K}_5,\boldsymbol{K}_6,\boldsymbol{K}_7,\boldsymbol{K}_0,\boldsymbol{K}_1,\boldsymbol{K}_2,\boldsymbol{K}_3,\boldsymbol{K}_4,\boldsymbol{K}_5,\boldsymbol{K}_6,\boldsymbol{K}_7,\boldsymbol{K}_7,\boldsymbol{K}_6,\boldsymbol{K}_5,\boldsymbol{K}_4,\boldsymbol{K}_3,\boldsymbol{K}_2,\boldsymbol{K}_1,\boldsymbol{K}_0$.

2. Цикл расшифрования 32-Р:

$\boldsymbol{K}_0,\boldsymbol{K}_1,\boldsymbol{K}_2,\boldsymbol{K}_3,\boldsymbol{K}_4,\boldsymbol{K}_5,\boldsymbol{K}_6,\boldsymbol{K}_7,\boldsymbol{K}_7,\boldsymbol{K}_6,\boldsymbol{K}_5,\boldsymbol{K}_4,\boldsymbol{K}_3,\boldsymbol{K}_2,\boldsymbol{K}_1,\boldsymbol{K}_0,\boldsymbol{K}_7,\boldsymbol{K}_6,\boldsymbol{K}_5,\boldsymbol{K}_4,\boldsymbol{K}_3,\boldsymbol{K}_2,\boldsymbol{K}_1,\boldsymbol{K}_0,\boldsymbol{K}_7,\boldsymbol{K}_6,\boldsymbol{K}_5,\boldsymbol{K}_4,\boldsymbol{K}_3,\boldsymbol{K}_2,\boldsymbol{K}_1,\boldsymbol{K}_0$.

3. Цикл выработки имитовставки 16-З:

$\boldsymbol{K}_0,\boldsymbol{K}_1,\boldsymbol{K}_2,\boldsymbol{K}_3,\boldsymbol{K}_4,\boldsymbol{K}_5,\boldsymbol{K}_6,\boldsymbol{K}_7,\boldsymbol{K}_0,\boldsymbol{K}_1,\boldsymbol{K}_2,\boldsymbol{K}_3,\boldsymbol{K}_4,\boldsymbol{K}_5,\boldsymbol{K}_6,\boldsymbol{K}_7$.

Каждый из циклов имеет собственное буквенно-цифровое обозначение, соответствующее шаблону «*n*-*X*», где первый элемент обозначения (n), задает число повторений основного шага в цикле, а второй элемент обозначения (X), буква, задает порядок зашифрования («З») или расшифрования («Р») в использовании ключевых элементов. Этот порядок нуждается в дополнительном пояснении:

Цикл расшифрования должен быть обратным циклу зашифрования, то есть последовательное применение этих двух циклов к произвольному блоку должно дать в итоге исходный блок, что отражается следующим соотношением: $\boldsymbol{Ц}_{32\text{-Р}}(\boldsymbol{Ц}_{32\text{-З}}(\boldsymbol{T}))=\boldsymbol{T}$, где $\boldsymbol{T}$ – произвольный 64-битовый блок данных, $\boldsymbol{Ц}_{\mathbf{X}}(\boldsymbol{T})$ – результат выполнения цикла X над блоком данных $\boldsymbol{T}$. Для выполнения этого условия для алгоритмов, подобных ГОСТу, необходимо и достаточно, чтобы порядок использования ключевых элементов соответствующими циклами был взаимно обратным. В справедливости записанного условия для рассматриваемого случая легко убедиться, сравнив приведенные выше последовательности для циклов 32-З и 32-Р. Из сказанного вытекает одно интересное следствие: свойство цикла быть обратным другому циклу является взаимным, то есть цикл

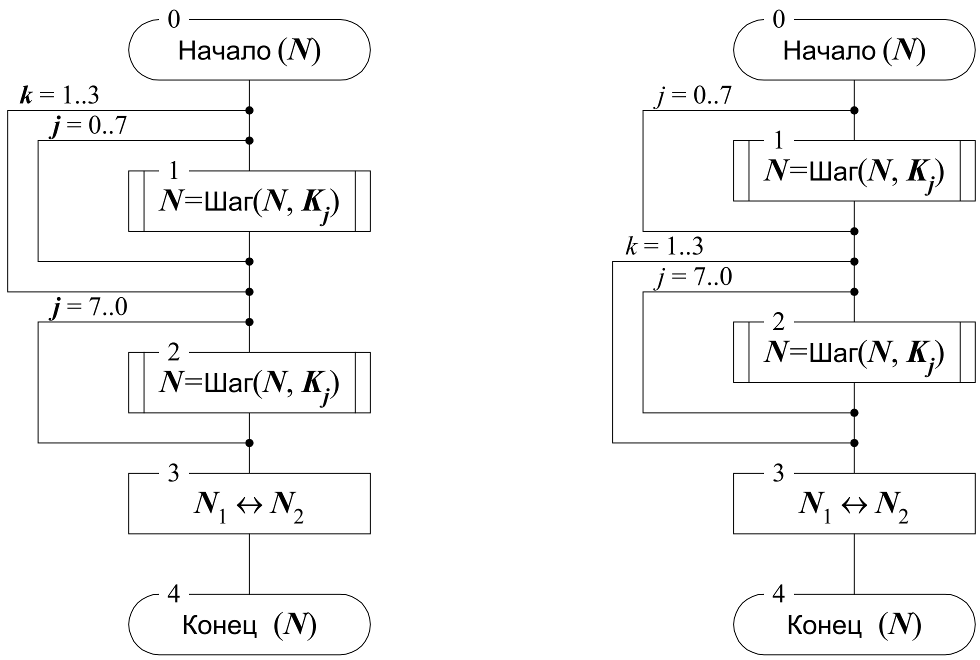

Рис. 2а. Схема цикла зашифрования 32-З.

Рис. 2б. Схема цикла расшифрования 32-Р.

32-З является обратным по отношению к циклу 32-Р. Другими словами, зашифрование блока данных теоретически может быть выполнено с помощью цикла расшифрования, в этом случае расшифрование блока данных должно быть выполнено циклом зашифрования. Из двух взаимно обратных циклов любой может быть использован для зашифрования, тогда второй должен быть использован для расшифрования данных, однако стандарт ГОСТ28147-89 закрепляет роли за циклами и не предоставляет пользователю права выбора в этом вопросе.

Цикл выработки имитовставки вдвое короче циклов шифрования, порядок использования ключевых элементов в нем такой же, как в первых 16 шагах цикла зашифрования, в чем нетрудно убедиться, рассмотрев приведенные выше последовательности, поэтому этот порядок в обозначении цикла кодируется той же самой буквой «З».

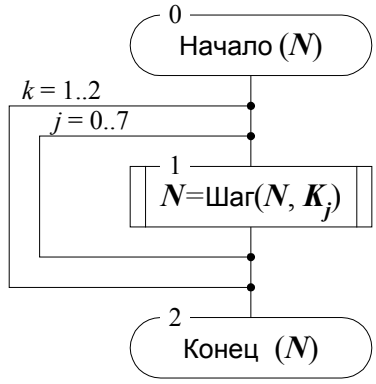

Рис. 2в. Схема цикла выработки имитовставки 16-З.

Схемы базовых циклов приведены на рисунках 2а-в. Каждый из них принимает в качестве аргумента и возвращает в качестве результата 64-битовый блок данных, обозначенный на схемах N. Символ Шаг(N,X) обозначает выполнение основного шага криптопреобразования для блока N с использованием ключевого элемента X. Между циклами шифрования и вычисления имитовставки есть еще одно отличие, не упомянутое выше: в конце базовых циклов шифрования старшая и младшая часть блока результата меняются местами, это необходимо для их взаимной обратимости.

1.5. Основные режимы шифрования.

ГОСТ 28147-89 предусматривает три следующих режима шифрования данных:
- простая замена,
- гаммирование,
- гаммирование с обратной связью,

и один дополнительный режим выработки имитовставки.

В любом из этих режимов данные обрабатываются блоками по 64 бита, на которые разбивается массив, подвергаемый криптографическому преобразованию, именно поэтому ГОСТ относится к блочным шифрам. Однако в двух режимах гаммирования есть возможность обработки неполного блока данных размером меньше 8 байт, что существенно при шифровании массивов данных с произвольным размером, который может быть не кратным 8 байтам.

Прежде чем перейти к рассмотрению конкретных алгоритмов криптографических преобразований, необходимо пояснить обозначения, используемые на схемах в следующих разделах:

$T_{\text{О}}, T_{\text{Ш}}$ – массивы соответственно открытых и зашифрованных данных;

$T_i^{\text{О}}, T_i^{\text{Ш}}$ – i-тые по порядку 64-битовые блоки соответственно открытых и зашифрованных данных: $T_{\text{О}} = (T_1^{\text{О}}, T_2^{\text{О}}, ..., T_n^{\text{О}})$, $T_{\text{Ш}} = (T_1^{\text{Ш}}, T_2^{\text{Ш}}, ..., T_n^{\text{Ш}})$, $1 \le i \le n$, последний блок может быть неполным: $|T_i^{\text{О}}| = |T_i^{\text{Ш}}| = 64$ при $1 \le i < n$, $1 \le |T_n^{\text{О}}| = |T_n^{\text{Ш}}| \le 64$;

n – число 64-битовых блоков в массиве данных;

$\boldsymbol{Ц}_{\mathbf{X}}$ – функция преобразования 64-битового блока данных по алгоритму базового цикла «X».

Теперь опишем основные режимы шифрования:

1. Простая замена.

Зашифрование в данном режиме заключается в применении цикла 32-З к блокам открытых данных, расшифрование – цикла 32-Р к блокам зашифрованных данных. Это наиболее простой из режимов, 64-битовые блоки данных обрабатываются в нем независимо друг от друга. Схемы алгоритмов зашифрования и расшифрования в режиме простой замены приведены на рисунках 3а и б соответственно, они тривиальны и не нуждаются в комментариях.

Размер массива открытых или зашифрованных данных, подвергающийся соответственно зашифрованию или расшифрованию, должен быть кратен 64 битам: $|\boldsymbol{T}_{\text{о}}| = |\boldsymbol{T}_{\text{ш}}| = 64 \cdot \boldsymbol{n}$, после выполнения операции размер полученного массива данных не изменяется.

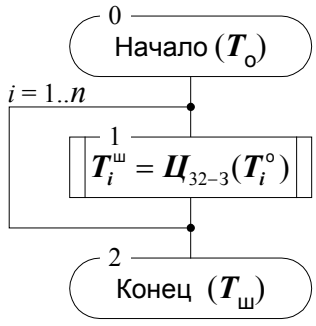

Рис. 3а. Алгоритм зашифрования данных в режиме простой замены.

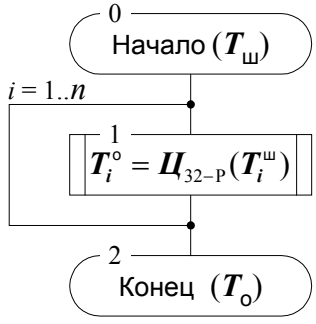

Рис. 3б. Алгоритм расшифрования данных в режиме простой замены.

Режим шифрования простой заменой имеет следующие особенности:

1. Так как блоки данных шифруются независимо друг от друга и от их позиции в массиве, при зашифровании двух одинаковых блоков открытого текста получаются одинаковые блоки шифртекста и наоборот. Отмеченное свойство позволит криптоаналитику сделать заключение о тождественности блоков исходных данных, если в массиве зашифрованных данных ему встретились идентичные блоки, что является недопустимым для серьезного шифра.
2. Если длина шифруемого массива данных не кратна 8 байтам или 64 битам, возникает проблема, чем и как дополнять последний неполный блок данных массива до полных 64 бит. Эта задача не так проста, как кажется на первый взгляд. Очевидные решения типа «дополнить неполный блок нулевыми битами» или, более обще, «дополнить неполный блок фиксированной комбинацией нулевых и единичных битов» могут при определенных условиях дать в руки криптоаналитика возможность методами перебора определить содержимое этого самого неполного блока, и этот факт означает снижение стойкости шифра. Кроме того, длина шифртекста при этом изменится, увеличившись до ближайшего целого, кратного 64 битам, что часто бывает нежелательным.

На первый взгляд, перечисленные выше особенности делают практически невозможным использование режима простой замены, ведь он может применяться только для шифрования массивов данных с размером кратным 64 битам, не содержащим повторяющихся 64-битовых блоков. Кажется, что для любых реальных данных гарантировать выполнение указанных условий невозможно. Это почти так, но есть одно очень важное исключение: вспомните, что размер ключа составляет 32 байта, а размер таблицы замен – 64 байта. Кроме того, наличие повторяющихся 8-байтовых блоков в ключе или таблице замен будет говорить об их весьма плохом качестве, поэтому в реальных ключевых элементах

такого повторения быть не может. Таким образом, мы выяснили, что режим простой замены вполне подходит для шифрования ключевой информации, тем более, что прочие режимы для этой цели менее удобны, поскольку требуют наличия дополнительного синхронизирующего элемента данных – синхропосылки (см. следующий раздел). Наша догадка верна, ГОСТ предписывает использовать режим простой замены исключительно для шифрования ключевых данных.

2. Гаммирование.

Как же можно избавиться от недостатков режима простой замены? Для этого необходимо сделать возможным шифрование блоков с размером менее 64 бит и обеспечить зависимость блока шифртекста от его номера, иными словами, ***рандомизировать*** процесс шифрования. В ГОСТе это достигается двумя различными способами в двух режимах шифрования, предусматривающих ***гаммирование***. ***Гаммирование*** – это наложение (снятие) на открытые (зашифрованные) данные криптографической гаммы, то есть последовательности элементов данных, вырабатываемых с помощью некоторого криптографического алгоритма, для получения зашифрованных (открытых) данных. Для наложения гаммы при зашифровании и ее снятия при расшифровании должны использоваться взаимно обратные бинарные операции, например, сложение и вычитание по модулю 2^{64} для 64-битовых блоков данных. В ГОСТе для этой цели используется операция побитного сложения по модулю 2, поскольку она является обратной самой себе и, к тому же, наиболее просто реализуется аппаратно. Гаммирование решает обе упомянутые проблемы; во первых, все элементы гаммы различны для реальных шифруемых массивов и, следовательно, результат зашифрования даже двух одинаковых блоков в одном массиве данных будет различным. Во вторых, хотя элементы гаммы и вырабатываются одинаковыми порциями в 64 бита, использоваться может и часть такого блока с размером, равным размеру шифруемого блока.

Теперь перейдем непосредственно к описанию режима гаммирования. Гамма для этого режима получается следующим образом: с помощью некоторого алгоритмического рекуррентного генератора последовательности чисел (РГПЧ) вырабатываются 64-битовые блоки данных, которые далее подвергаются преобразованию по циклу 32-З, то есть зашифрованию в режиме простой замены, в результате получаются блоки гаммы. Благодаря тому, что наложение и снятие гаммы осуществляется при помощи одной и той же операции побитового исключающего или, алгоритмы зашифрования и расшифрования в режиме гаммирования идентичны, их общая схема приведена на рисунке 5.

РГПЧ, используемый для выработки гаммы, является рекуррентной функцией: $\boldsymbol{\Omega}_{i+1} = \boldsymbol{f}(\boldsymbol{\Omega}_i)$, где $\boldsymbol{\Omega}_i$ – элементы рекуррентной последовательности, $\boldsymbol{f}$ – функция преобразования. Следовательно, неизбежно возникает вопрос о его инициализации, то есть об элементе $\boldsymbol{\Omega}_0$. В действительности, этот элемент данных является параметром алгоритма для режимов гаммирования, на схемах он обозначен как $\boldsymbol{S}$, и называется в криптографии ***синхропосылкой***, а в нашем ГОСТе – ***начальным заполнением*** одного из регистров шифратора. По определенным соображениям разработчики ГОСТа решили использовать для инициализации РГПЧ не непосредственно синхропосылку, а результат ее преобразования по циклу 32-З: $\boldsymbol{\Omega}_0 = \boldsymbol{Ц}_{32\text{-}З}(\boldsymbol{S})$. Последовательность элементов, вырабатываемых РГПЧ, целиком зависит от его начального заполнения, то есть элементы этой последовательности являются функцией своего номера и начального заполнения РГПЧ: $\boldsymbol{\Omega}_i = \boldsymbol{f}_i(\boldsymbol{\Omega}_0)$, где $\boldsymbol{f}_i(\boldsymbol{X}) = \boldsymbol{f}(\boldsymbol{f}_{i-1}(\boldsymbol{X}))$, $\boldsymbol{f}_0(\boldsymbol{X}) = \boldsymbol{X}$. С учетом преобразования по алгоритму простой замены добавляется еще и зависимость от ключа:
$\boldsymbol{Г}_i = \boldsymbol{Ц}_{32\text{-}З}(\boldsymbol{\Omega}_i) = \boldsymbol{Ц}_{32\text{-}З}(\boldsymbol{f}_i(\boldsymbol{\Omega}_0)) = \boldsymbol{Ц}_{32\text{-}З}(\boldsymbol{f}_i(\boldsymbol{Ц}_{32\text{-}З}(\boldsymbol{S}))) = \varphi_i(\boldsymbol{S},\boldsymbol{K})$, где $\boldsymbol{Г}_i$ – $\boldsymbol{i}$-тый элемент гаммы, $\boldsymbol{K}$ – ключ.

Таким образом, последовательность элементов гаммы для использования в режиме гаммирования однозначно определяется ключевыми данными и синхропосылкой. Естественно, для обратимости процедуры шифрования в процессах за- и расшифрования должна использоваться одна и та же синхропосылка. Из требования уникальности гаммы, невыполнение которого приводит к катастрофическому снижению стойкости шифра, следует, что для шифрования двух различных массивов данных на одном ключе необходимо обеспечить использование различных синхропосылок. Это приводит к необходимости хранить или передавать синхропосылку по каналам связи вместе с зашифрованными данными, хотя в отдельных особых случаях она может быть предопределена или вычисляться особым образом, если исключается шифрование двух массивов на одном ключе.

Теперь подробно рассмотрим РГПЧ, используемый в ГОСТе для генерации элементов гаммы. Прежде всего, надо отметить, что к нему не предъявляются требования обеспечения каких-либо статистических характеристик вырабатываемой последовательности чисел. РГПЧ спроектирован разработчиками ГОСТа исходя из необходимости выполнения следующих условий:

- период повторения последовательности чисел, вырабатываемой РГПЧ, не должен сильно (в процентном отношении) отличаться от максимально возможного при заданном размере блока значения 2^{64};
- соседние значения, вырабатываемые РГПЧ, должны отличаться друг от друга в каждом байте, иначе задача криптоаналитика будет упрощена;
- РГПЧ должен быть достаточно просто реализуем как аппаратно, так и программно на наиболее распространенных типах процессоров, большинство из которых, как известно, имеют разрядность 32 бита.

Исходя из перечисленных принципов создатели ГОСТа спроектировали весьма удачный РГПЧ, имеющий следующие характеристики:

- в 64-битовом блоке старшая и младшая части обрабатываются независимо друг от друга: $\boldsymbol{\Omega_i} = (\boldsymbol{\Omega_i^0}, \boldsymbol{\Omega_i^1})$, $|\boldsymbol{\Omega_i^0}| = |\boldsymbol{\Omega_i^1}| = 32$, $\boldsymbol{\Omega_{i+1}^0} = \hat{\boldsymbol{f}}(\boldsymbol{\Omega_i^0})$, $\boldsymbol{\Omega_{i+1}^1} = \tilde{\boldsymbol{f}}(\boldsymbol{\Omega_i^1})$; фактически, существуют два независимых РГПЧ для старшей и младшей частей блока.
- рекуррентные соотношения для старшей и младшей частей следующие:
  $\boldsymbol{\Omega_{i+1}^0} = (\boldsymbol{\Omega_i^0} + \boldsymbol{C}_1) mod 2^{32}$, где $\boldsymbol{C}_1 = 1010101_{16}$;
  $\boldsymbol{\Omega_{i+1}^1} = (\boldsymbol{\Omega_i^1} + \boldsymbol{C}_2 - 1) mod (2^{32} - 1) + 1$, где $\boldsymbol{C}_2 = 1010104_{16}$;
  Нижний индекс в записи числа означает его систему счисления, таким образом, константы, используемые на данном шаге, записаны в 16-ричной системе счисления. Второе выражение нуждается в комментариях, так как в тексте ГОСТа приведено нечто другое: $\boldsymbol{\Omega_{i+1}^1} = (\boldsymbol{\Omega_i^1} + \boldsymbol{C}_2) mod (2^{32} - 1)$, с тем же значением константы $\boldsymbol{C}_2$. Но далее в тексте стандарта дается комментарий, что, оказывается, под операцией взятия остатка по модулю 2^{32}–1 *там* понимается не то же самое, что и в математике. Отличие заключается в том, что согласно ГОСТу $(2^{32}–1) \bmod (2^{32}–1) = (2^{32}–1)$, а не 0. На самом деле, это упрощает реализацию формулы, а математически корректное выражение для нее приведено выше.
- период повторения последовательности для младшей части составляет 2^{32}, для старшей части 2^{32}–1, для всей последовательности период составляет $2^{32} \cdot (2^{32}–1)$, доказательство этого факта, весьма несложное, получите сами. Первая формула из двух реализуется за одну команду, вторая, несмотря на ее кажущуюся громоздкость, за две команды на всех современных 32-разрядных процессорах.

Схема алгоритма шифрования в режиме гаммирования приведена на рисунке 4, ниже изложены пояснения к схеме:

Шаг 0. Определяет исходные данные для основного шага криптопреобразования:

- $\boldsymbol{T}_{\text{о(ш)}}$ – массив открытых (зашифрованных) данных произвольного размера, подвергаемый процедуре зашифрования (расшифрования), по ходу процедуры массив подвергается преобразованию порциями по 64 бита;
- $\boldsymbol{S}$ – ***синхропосылка***, 64-битовый элемент данных, необходимый для инициализации генератора гаммы;

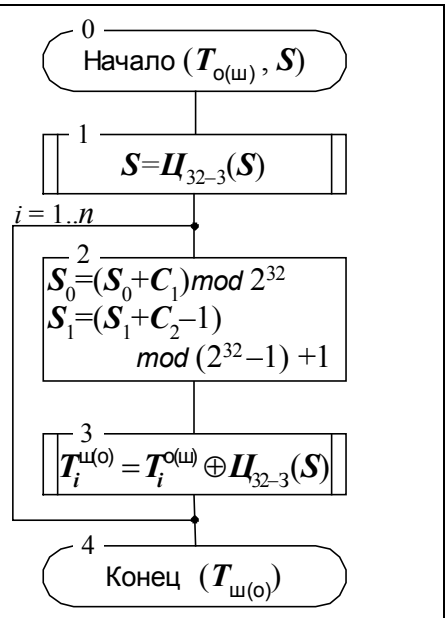

Рис. 4. Алгоритм зашифрования (расшифрования) данных в режиме гаммирования.

Шаг 1. Начальное преобразование синхропосылки, выполняемое для ее «рандомизации», то есть для устранения статистических закономерностей, присутствующих в ней, результат используется как начальное заполнение РГПЧ;

Шаг 2. Один шаг работы РГПЧ, реализующий его рекуррентный алгоритм. В ходе данного шага старшая ($\boldsymbol{S}_1$) и младшая ($\boldsymbol{S}_0$) части последовательности данных вырабатываются независимо друг от друга;

Шаг 3. Гаммирование. Очередной 64-битовый элемент, выработанный РГПЧ, подвергается процедуре зашифрования по циклу 32–З, результат используется как элемент гаммы для зашифрования (расшифрования) очередного блока открытых (зашифрованных) данных того же размера.

Шаг 4. Результат работы алгоритма – зашифрованный (расшифрованный) массив данных.

Ниже перечислены особенности гаммирования как режима шифрования.

1. Одинаковые блоки в открытом массиве данных дадут при зашифровании различные блоки шифртекста, что позволит скрыть факт их идентичности.
2. Поскольку наложение гаммы выполняется побитно, шифрование неполного блока данных легко выполнимо как шифрование битов этого неполного блока, для чего используется соответствующие биты блока гаммы . Так, для зашифрования неполного блока в 1 бит можно использовать любой бит из блока гаммы.
3. Синхропосылка, использованная при зашифровании, каким-то образом должна быть передана для использования при расшифровании. Это может быть достигнуто следующими путями:
   - хранить или передавать синхропосылку вместе с зашифрованным массивом данных, что приведет к увеличению размера массива данных при зашифровании на размер синхропосылки, то есть на 8 байт;
   - использовать предопределенное значение синхропосылки или вырабатывать ее синхронно источником и приемником по определенному закону, в этом случае изменение размера передаваемого или хранимого массива данных отсутствует;

Оба способа дополняют друг друга, и в тех редких случаях, где не работает первый, наиболее употребительный из них, может быть использован второй, более экзотический. Второй способ имеет гораздо меньшее применение, поскольку сделать синхропосылку предопределенной можно только в том случае, если на данном комплекте ключевой информации шифруется заведомо не более одного массива данных, что бывает в редких случаях. Генерировать синхропосылку синхронно у источника и получателя массива данных также не всегда представляется возможным, поскольку требует жесткой привязки к чему-либо в системе. Так, здравая на первый взгляд идея использовать в качестве синхропосылки в системе передачи зашифрованных сообщений номер передаваемого сообщения не

подходит, поскольку сообщение может потеряться и не дойти до адресата, в этом случае произойдет десинхронизация систем шифрования источника и приемника. Поэтому в рассмотренном случае нет альтернативы передаче синхропосылки вместе с зашифрованным сообщением.

С другой стороны, можно привести и обратный пример. Допустим, шифрование данных используется для защиты информации на диске, и реализовано оно на низком уровне, для обеспечения независимого доступа данные шифруются по секторам. В этом случае невозможно хранить синхропосылку вместе с зашифрованными данными, поскольку размер сектора нельзя изменить, однако ее можно вычислять как некоторую функцию от номера считывающей головки диска, номера дорожки (цилиндра) и номера сектора на дорожке. В этом случае синхропосылка привязывается к положению сектора на диске, которое вряд ли может измениться без переформатирования диска, то есть без уничтожения данных на нем.

Режим гаммирования имеет еще одну интересную особенность. В этом режиме биты массива данных шифруются независимо друг от друга. Таким образом, каждый бит шифртекста зависит от соответствующего бита открытого текста и, естественно, порядкового номера бита в массиве: $t_i^{\text{ш}} = t_i^{\text{о}} \oplus \gamma_i = f(t_i^{\text{о}}, i)$. Из этого вытекает, что изменение бита шифртекста на противоположное значение приведет к аналогичному изменению бита открытого текста на противоположный:

$$\overline{t}_i^{\text{ш}} = t_i^{\text{ш}} \oplus 1 = (t_i^{\text{о}} \oplus \gamma_i) \oplus 1 = (t_i^{\text{о}} \oplus 1) \oplus \gamma_i = \overline{t}_i^{\text{о}} \oplus \gamma_i,$$

где $\bar{t}$ обозначает инвертированное по отношению к t значение бита ($\overline{0} = 1$, $\overline{1} = 0$).

Данное свойство дает злоумышленнику возможность воздействуя на биты шифртекста вносить предсказуемые и даже целенаправленные изменения в соответствующий открытый текст, получаемый после его расшифрования, не обладая при этом секретным ключом. Это иллюстрирует хорошо известный в криптологии факт, что ***секретность и аутентичность суть различные свойства*** криптографических систем. Иными словами, свойства криптосистемы обеспечивать защиту от несанкционированного ознакомления с содержимым сообщения и от несанкционированного внесения изменений в сообщение являются независимыми и лишь в отдельных случаях могут пересекаться. Сказанное означает, что существуют криптографические алгоритмы, обеспечивающие определенную секретность зашифрованных данных и при этом никак не защищающие от внесения изменений и наоборот, обеспечивающие аутентичность данных и никак не ограничивающие возможность ознакомления с ними. По этой причине рассматриваемое свойство режима гаммирования не должно рассматриваться как его недостаток.

3. Гаммирование с обратной связью.

Данный режим очень похож на режим гаммирования и отличается от него только способом выработки элементов гаммы – очередной элемент гаммы вырабатывается как результат преобразования по циклу 32-З предыдущего блока зашифрованных данных, а для зашифрования первого блока массива данных элемент гаммы вырабатывается как результат преобразования по тому же циклу синхропосылки. Этим достигается зацепление блоков – каждый блок шифртекста в этом режиме зависит от соответствующего и всех предыдущих блоков открытого текста. Поэтому данный режим иногда называется ***гаммированием с зацеплением блоков***. На стойкость шифра факт зацепления блоков не оказывает никакого влияния.

Схема алгоритмов за- и расшифрования в режиме гаммирования с обратной связью приведена на рисунке 5 и ввиду своей простоты в комментариях не нуждается.

Шифрование в режиме гаммирования с обратной связью обладает теми же особенностями, что и шифрование в режиме обычного гаммирования, за исключением влияния искажений шифртекста на соответствующий открытый текст. Для сравнения запишем функции расшифрования блока для обоих упомянутых режимов:

$\boldsymbol{T}_i^{\circ} = \boldsymbol{T}_i^{\text{ш}} \oplus \Gamma_i$, гаммирование;

$\boldsymbol{T}_i^{\circ} = \boldsymbol{T}_i^{\text{ш}} \oplus \boldsymbol{U}_{32-3}(\boldsymbol{T}_{i-1}^{\text{ш}})$, гаммирование с обратной связью;

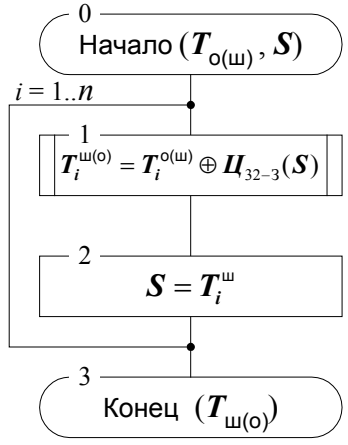

Рис. 5. Алгоритм зашифрования (расшифрования) данных в режиме гаммирования с обратной связью.

Если в режиме обычного гаммирования изменения в определенных битах шифртекста влияют только на соответствующие биты открытого текста, то в режиме гаммирования с обратной связью картина несколько сложнее. Как видно из соответствующего уравнения, при расшифровании блока данных в режиме гаммирования с обратной связью, блок открытых данных зависит от соответствующего и предыдущего блоков зашифрованных данных. Поэтому, если внести искажения в зашифрованный блок, то после расшифрования искаженными окажутся два блока открытых данных – соответствующий и следующий за ним, причем искажения в первом случае будут носить тот же характер, что и в режиме гаммирования, а во втором случае – как в режиме простой замены. Другими словами, в соответствующем блоке открытых данных искаженными окажутся те же самые биты, что и в блоке шифрованных данных, а в следующем блоке открытых данных все биты независимо друг от друга с вероятностью 1⁄2 изменят свои значения.

4. Выработка имитовставки к массиву данных.

В предыдущих разделах мы обсудили влияние искажения шифрованных данных на соответствующие открытые данные. Мы установили, что при расшифровании в режиме простой замены соответствующий блок открытых данных оказывается искаженным непредсказуемым образом, а при расшифровании блока в режиме гаммирования изменения предсказуемы. В режиме гаммирования с обратной связью искаженными оказываются два блока, один предсказуемым, а другой непредсказуемым образом. Значит ли это, что с точки зрения защиты от навязывания ложных данных режим гаммирования является плохим, а режимы простой замены и гаммирования с обратной связью хорошими? – Ни в коем случае. При анализе данной ситуации необходимо учесть то, что непредсказуемые изменения в расшифрованном блоке данных могут быть обнаружены только в случае избыточности этих самых данных, причем чем больше степень избыточности, тем вероятнее обнаружение искажения. Очень большая избыточность имеет место, например, для текстов на естественных и искусственных языках, в этом случае факт искажения обнаруживается практически неизбежно. Однако в других случаях, например, при искажении сжатых звуковых образов, мы получим просто другой образ, который сможет воспринять наше ухо. Искажение в этом случае останется необнаруженным, если, конечно, нет априорной информации о характере звука. Вывод здесь такой: поскольку способность некоторых режимов шифрования обнаруживать искажения, внесенные в шифрованные данные, существенным образом опирается на наличие и степень избыточности шифруемых данных, эта способность не является имманентным свойством соответствующих режимов и не может рассматриваться как их достоинство.

Для решения задачи обнаружения искажений в зашифрованном массиве данных с заданной вероятностью в ГОСТе предусмотрен дополнительный режим криптографического

преобразования – выработка имитовставки. Имитовставка – это контрольная комбинация, зависящая от открытых данных и секретной ключевой информации. Целью использования имитовставки является обнаружение всех случайных или преднамеренных изменений в массиве информации. Проблема, изложенная в предыдущем пункте, может быть успешно решена с помощью добавления к шифрованным данным имитовставки. Для потенциального злоумышленника две следующие задачи практически неразрешимы, если он не владеет ключевой информацией:

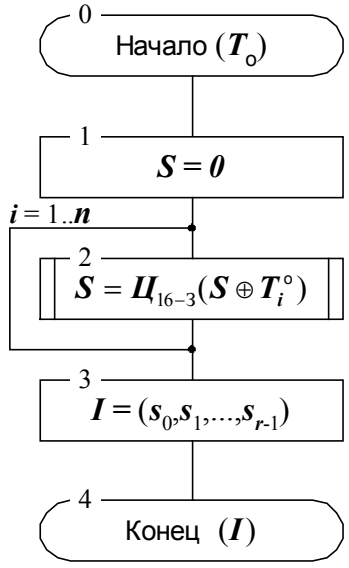

Рис. 6. Алгоритм выработки имитовставки для массива данных.

- вычисление имитовставки для заданного открытого массива информации;
- подбор открытых данных под заданную имитовставку;

Схема алгоритма выработки имитовставки приведена на рисунке 6. В качестве имитовставки берется часть блока, полученного на выходе, обычно – 32 его младших бита. При выборе размера имитовставки надо принимать во внимание, что вероятность успешного навязывания ложных данных равна величине $2^{-|I|}$ на одну попытку подбора, если в распоряжении злоумышленника нет более эффективного метода подбора, чем простое угадывание. При использовании имитовставки размером 32 бита эта вероятность равна $2^{-32} \approx 0.23 \cdot 10^{-9}$.

2. Обсуждение криптографических алгоритмов ГОСТа.

2.1. Криптографическая стойкость ГОСТа.

При выборе криптографического алгоритма для использования в конкретной разработке одним из определяющих факторов является его стойкость, то есть устойчивость к попыткам противоположной стороны его раскрыть. Вопрос о стойкости шифра при ближайшем рассмотрении сводится к двум взаимосвязанным вопросам:

- можно ли вообще раскрыть данный шифр;
- если да, то насколько это трудно сделать практически;

Шифры, которые вообще невозможно раскрыть, называются абсолютно или теоретически стойкими. Существование подобных шифров доказывается теоремой Шеннона, однако ценой этой стойкости является необходимость использования для шифрования каждого сообщения ключа, не меньшего по размеру самого сообщения. Во всех случаях за исключением ряда особых эта цена чрезмерна, поэтому на практике в основном используются шифры, не обладающие абсолютной стойкостью. Таким образом, наиболее употребительные схемы шифрования могут быть раскрыты за конечное время или, что точнее, за конечное число шагов, каждый из которых является некоторой операцией над числами. Для них наиважнейшее значение имеет понятие практической стойкости, выражающее практическую трудность их раскрытия. Количественной мерой этой трудности может служить число элементарных арифметических и логических операций, которые необходимо выполнить, чтобы раскрыть шифр, то есть чтобы для заданного шифртекста с вероятностью, не меньшей заданной величины, определить соответствующий открытый

текст. При этом в дополнении к дешифруемому массиву данных криптоаналитик может располагать блоками открытых данных и соответствующих им зашифрованных данных или даже возможностью получить для любых выбранных им открытых данных соответствующие зашифрованные данные – в зависимости от перечисленных и многих других неуказанных условий различают отдельные виды криптоанализа.

Все современные криптосистемы построены по принципу Кирхгоффа, то есть секретность зашифрованных сообщений определяется секретностью ключа. Это значит, что даже если сам алгоритм шифрования известен криптоаналитику, тот тем не менее не в состоянии расшифровать сообщение, если не располагает соответствующим ключом. Все классические блочные шифры, в том числе DES и ГОСТ, соответствуют этому принципу и спроектированы таким образом, чтобы не было пути вскрыть их более эффективным способом, чем полным перебором по всему ключевому пространству, т.е. по всем возможным значениям ключа. Ясно, что стойкость таких шифров определяется размером используемого в них ключа.

В шифре ГОСТ используется 256-битовый ключ и объем ключевого пространства составляет 2^{256}. Ни на одной из существующих в настоящее время или предполагаемых к реализации в недалеком будущем ЭВМ общего применения нельзя подобрать ключ за время, меньшее многих сотен лет. Российский стандарт проектировался с большим запасом и по стойкости на много порядков превосходит американский стандарт DES с его реальным размером ключа в 56 бит и объемом ключевого пространства всего 2^{56}. В свете прогресса современных вычислительных средств этого явно недостаточно. В этой связи DES может представлять скорее исследовательский или научный, чем практический интерес. Как ожидается, в 1998 году он перестанет быть стандартом США на шифрование.

2.2. Замечания по архитектуре ГОСТа.

Общеизвестно, что шифр ГОСТ 28147-89 является представителем целого семейства шифров, построенных на одних и тех же принципах. Самым известным его «родственником» является американский стандарт шифрования, алгоритм DES. Все эти шифры, подобно ГОСТу, содержат алгоритмы трех уровней. В основе всегда лежит некий «основной шаг», на базе которого сходным образом строятся «базовые циклы», и уже на их основе построены практические процедуры шифрования и выработки имитовставки. Таким образом, специфика каждого из шифров этого семейства заключена именно в его основном шаге, точнее даже в его части. Хотя архитектура классических блочных шифров, к которым относится ГОСТ, лежит далеко за пределами темы настоящей статьи, все же стоит сказать несколько слов по ее поводу.

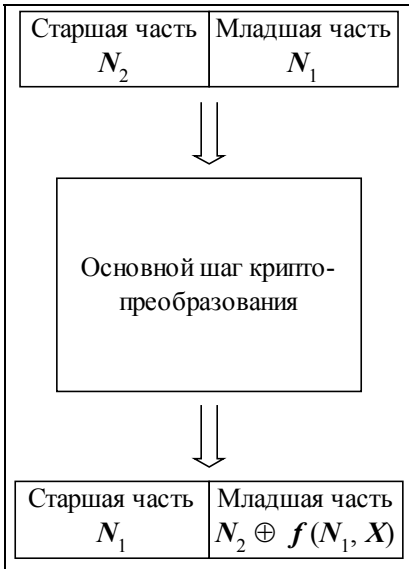

Рис. 7. Содержание основного шага криптопреобразования для блочных шифров, подобных ГОСТу.

Алгоритмы «основных шагов криптопреобразования» для шифров, подобных ГОСТу, построены идентичным образом. Их общая схема приведена на рисунке 7. На вход основного шага подается блок четного размера, старшая и младшая половины которого обрабатываются отдельно друг от друга. В ходе преобразования младшая половина блока помещается на место старшей, а старшая, скомбинированная с помощью операции побитового ***исключающего или*** с результатом вычисления некоторой функции, на место

младшей. Эта функция, принимающая в качестве аргумента младшую половину блока и некоторый элемент ключевой информации (X), является содержательной частью шифра и называется его ***функцией шифрования***. Соображения стойкости шифра требуют, чтобы размеры всех перечисленных элементов блоков были равны: $|\boldsymbol{N}_1| = |\boldsymbol{N}_2| = |\boldsymbol{X}|$, в ГОСТ и DESe они равны 32 битам.

Если применить сказанное к схеме основного шага алгоритма ГОСТ, станет очевидным, что блоки 1,2,3 (см. рис. 1) алгоритма определяют вычисление его функции шифрования, а блоки 4 и 5 задают формирование выходного блока основного шага исходя из содержимого входного блока и значения функции шифрования.

В предыдущем разделе мы уже сравнили DES и ГОСТ по стойкости, теперь мы сравним их по функциональному содержанию и удобству реализации. В циклах шифрования ГОСТа основной шаг повторяется 32 раза, для DESa эта величина равна 16. Однако сама функция шифрования ГОСТа существенно проще аналогичной функции DES, в которой присутствует множество нерегулярных битовых перестановок. Эти операции чрезвычайно неэффективно реализуются на современных неспециализированных процессорах. ГОСТ не содержит подобных операций, поэтому он значительно удобней для программной реализации.

Ни одна из рассмотренных автором реализаций DESa для платформы Intel x86 не достигает даже половины производительности предложенной вашему вниманию в настоящей статье реализации ГОСТа, несмотря на вдвое более короткий цикл. Все сказанное выше свидетельствует о том, что разработчики ГОСТа учли как положительные, так и отрицательные стороны DESa, а также более реально оценили текущие и перспективные возможности криптоанализа.

2.3. Требования к качеству ключевой информации и источники ключей.

Не все ключи и таблицы замен обеспечивают максимальную стойкость шифра. Для каждого алгоритма шифрования существуют свои критерии оценки ключевой информации. Так, для алгоритма DES известно существование так называемых «***слабых ключей***», при использовании которых связь между открытыми и зашифрованными данными не маскируется достаточным образом, и шифр сравнительно просто вскрывается.

Исчерпывающий ответ на вопрос о критериях качества ключей и таблиц замен ГОСТа если и можно вообще где-либо получить, то только у разработчиков алгоритма. Соответствующие данные не были опубликованы в открытой печати. Однако согласно установленному порядку, для шифрования информации, имеющей гриф, должны быть использованы ключевые данные, полученные от уполномоченной организации. Косвенным образом это может свидетельствовать о наличии методик проверки ключевых данных на «вшивость». Сам факт существования слабых ключевых данных в Российском стандарте шифрования не вызывает сомнения. Очевидно, нулевой ключ и тривиальная таблица замен, по которой любое значение заменяется но него самого, являются слабыми, при использовании хотя бы одного из них шифр достаточно просто взламывается, каков бы ни был второй ключевой элемент.

Как уже было отмечено выше, критерии оценки ключевой информации недоступны, однако на их счет все же можно высказать некоторые общие соображения:

1. ***Ключ*** должен являться массивом статистически независимых битов, принимающих с равной вероятностью значения 0 и 1. При этом некоторые конкретные значения ключа могут оказаться «слабыми», то есть шифр может не обеспечивать заданный уровень стойкости в случае их использования. Однако, предположительно, доля таких значений в общей массе всех возможных ключей ничтожно мала. Поэтому ключи, выработанные с

помощью некоторого датчика истинно случайных чисел, будут качественными с вероятностью, отличающейся от единицы на ничтожно малую величину. Если же ключи вырабатываются с помощью генератора псевдослучайных чисел, то используемый генератор должен обеспечивать указанные выше статистические характеристики, и, кроме того, обладать высокой криптостойкостью, не меньшей, чем у самого ГОСТа. Иными словами, задача определения отсутствующих членов вырабатываемой генератором последовательности элементов не должна быть проще, чем задача вскрытия шифра. Кроме того, для отбраковки ключей с плохими статистическими характеристиками могут быть использованы различные статистические критерии. На практике обычно хватает двух критериев, – для проверки равновероятного распределения битов ключа между значениями 0 и 1 обычно используется критерий Пирсона («хи квадрат»), а для проверки независимости битов ключа – критерий серий. Об упомянутых критериях можно прочитать в учебниках или справочниках по математической статистике.

2. ***Таблица замен*** является долговременным ключевым элементом, то есть действует в течение гораздо более длительного срока, чем отдельный ключ. Предполагается, что она является общей для всех узлов шифрования в рамках одной системы криптографической защиты. Даже при нарушении конфиденциальности таблицы замен стойкость шифра остается чрезвычайно высокой и не снижается ниже допустимого предела. К качеству отдельных узлов замен можно предъявить приведенное ниже требование. Каждый узел замен может быть описан четверкой логических функций, каждая из которых имеет четыре логических аргумента. Необходимо, чтобы эти функции были достаточно сложными. Это требование сложности невозможно выразить формально, однако в качестве необходимого условия можно потребовать, чтобы соответствующие логические функции, записанные в минимальной форме (т.е. с минимально возможной длиной выражения) с использованием основных логических операций, не были короче некоторого необходимого минимума. В первом и очень грубом приближении это условие может сойти и за достаточное. Кроме того, отдельные функции в пределах всей таблицы замен должны отличаться друг от друга в достаточной степени. На практике бывает достаточно получить узлы замен как независимые случайные перестановки чисел от 0 до 15, это может быть практически реализовано, например, с помощью перемешивания колоды из шестнадцати карт, за каждой из которых закреплено одно из значений указанного диапазона.

Необходимо отметить еще один интересный факт относительно таблицы замен. Для обратимости циклов шифрования «32-З» и «32-Р» не требуется, чтобы узлы замен были перестановками чисел от 0 до 15. Все работает даже в том случае, если в узле замен есть повторяющиеся элементы, и замена, определяемая таким узлом, необратима, однако в этом случае снижается стойкость шифра. Почему это именно так, не рассматривается в настоящей статье, однако в самом факте убедиться несложно. Для этого достаточно, используя демонстрационную программу шифрования файлов данных, прилагающуюся к настоящей статье, зашифровать а затем расшифровать файл данных, использовав для этой процедуры «неполноценную» таблицу замен, узлы которой содержат повторяющиеся значения.

Если вы разрабатываете программы, использующие криптографические алгоритмы, вам необходимо позаботиться об утилитах, вырабатывающих ключевую информацию, а для таких утилит необходим источник случайных чисел (СЧ) высокого статистического качества и криптостойкости. Наилучшим подходом здесь было бы использование аппаратных датчиков СЧ, однако это не всегда приемлемо по экономическим соображениям. В качестве разумной альтернативы возможно (и очень широко распространено) использование различных программных датчиков СЧ. При генерации небольшого по объему массива ключевой информации широко применяется метод «электронной рулетки», когда очередная

получаемая с такого датчика порция случайных битов зависит от момента времени нажатия оператором некоторой клавиши на клавиатуре компьютера.

Этот подход использован в программе генерации одного ключа, исходный текст которой на языке Си с ассемблерными вкраплениями прилагается к настоящей статье в файле make1key.c. Для выработки случайных чисел из заданного диапазона используется канал 2 системного таймера, информация считывается с него при нажатии оператором какой-либо клавиши на клавиатуре дисплея. За одно нажатие генерируется один байт ключа и на экран выводится точка. Чтобы было невозможно генерировать байты ключа удержанием клавиши в нажатом состоянии, между циклами генерации введена временная задержка и в начале каждого цикла проверяется, было ли во время паузы нажатие клавиши. Если таковое имело место, выдается звуковой сигнал и нажатие игнорируется. Программу целесообразно запускать только из «голого» DOSa, в DOS-сеансе Windows 3.x/95 она также работает, но нет уверенности в обеспечении нужных статистических характеристик, а под Windows NT программа по вполне понятным причинам (лезет напрямую в порты) вообще не работает корректно.

3. Замечания по реализации.

3.1. Три шага оптимизации.

Целью описываемой в настоящей статье реализации ГОСТа было не создание максимального по эффективности кода любой ценой, целью было создание близкого к оптимальному по быстродействию, но при этом компактного и легкого для восприятия программистом кода.

Программирование алгоритмов ГОСТа «в лоб» даже на языке ассемблера дает результат, очень далекий от возможного оптимума. Задача создания эффективной реализации указанных алгоритмов для 16-разрядных процессоров Intel 8088–80286 представляет собой хотя и не сверхсложную, но все же и не вполне тривиальную задачу, и требует очень хорошего знания архитектуры и системы команд упомянутого семейства процессоров. Трудность здесь заключается в том, что для достижения максимальной производительности программная реализация должна как можно большую часть операций с данными выполнять в регистрах и как можно реже обращаться к памяти, а регистров у рассматриваемых процессоров не так много и все они 16-разрядные. С 32-разрядными процессорами этой же линии (Intel 80386 и старше) все намного проще именно в силу их 32-разрядности, здесь не будет трудностей даже у новичка.

В реализации алгоритмов были использованы изложенные ниже подходы, позволившие достигнуть максимальной производительности. Первые два из них достаточно очевидны, настолько, что встречаются практически в каждой реализации ГОСТа.

1. Базовые циклы ГОСТа содержат вложенные циклы (звучит коряво, но по-другому не скажешь), причем во внутреннем цикле порядок использования восьми 32-битовых элементов ключа может быть прямой или обратный. Существенно упростить реализацию и повысить эффективность базовых циклов можно, если избежать использования вложенных циклов и просматривать последовательность элементов ключа только один раз. Для этого необходимо предварительно сформировать последовательность элементов ключа в том порядке, в котором они используются в соответствующем базовом цикле.

2. В основном шаге криптопреобразования 8 раз выполняется подстановка 4-битовых групп данных. Целевой процессор реализации не имеет команды замены 4-битовых групп, однако имеет удобную команду байтовой замены (**xlat**). Ее использование дает следующие выгоды:

- за одну команду выполняются сразу две замены;

- исчезает необходимость выделять полубайты из двойных слов для выполнения замены, а затем из 4-битовых результатов замен вновь формировать двойное слово.

Этим достигается значительное увеличение быстродействия кода, однако мир устроен так, что за все приходится платить, и в данном случае платой является необходимость преобразования таблицы замен. Каждая из четырех пар 4-разрядных узлов замен заменяется одним 8-разрядным узлом, который, говоря языком математики, представляет собой прямое произведение узлов, входящих в пару. Пара 4-разрядных узлов требует для своего представления 16 байтов, один 8-разрядный – 256 байтов. Таким образом, размер таблицы замен, которая должна храниться в памяти компьютера, увеличивается до $4 \cdot 256 = 1024$ байтов, или до одного килобайта. Конечно, такая плата за существенное увеличение эффективности реализации вполне приемлема.

3. После выполнения подстановок кода по таблице замен основной шаг криптопреобразования предполагает циклический сдвиг двойного слова влево на 11 бит. В силу 16-разрядной архитектуры рассматриваемых процессоров вращение 32-разрядного блока даже на 1 бит невозможно реализовать менее, чем за три ассемблерные команды, а вращение на большее число разрядов только как последовательность отдельных вращений на 1 разряд. К счастью, вращение на 11 бит влево можно представить как вращение на 8 бит, а затем еще на 3 бита влево. Думаю, для всех очевидно, что первое вращение реализуется тремя командами обмена байтовых регистров (**xchg**). Но секрет третьей оптимизации даже не в этом. Замена одного байта по таблице замен осуществляется командой **xlat**, которая выполняет операцию над аргументом в регистре **AL**, для того, чтобы заменить все байты двойного слова, их надо последовательно помещать в этот регистр. Секрет третьей оптимизации заключается в том, что эти перестановки можно организовать так, что в результате двойное слово окажется повернутым на 8 бит влево, то есть в совмещении замены по таблице и во вращении на байт влево. Еще один момент, на который стоит обратить внимание, это оптимальное кодирование трех последовательных вращений на 1 бит, это – может быть реализовано по-разному и важно было выбрать оптимальный способ. Оптимум оказался неочевидным, поскольку потребовал выхода за пределы логики битовых сдвигов и использования команды суммирования с битами переноса (**adc**), то есть бит помещается на свою позицию не командой сдвига, а командой суммирования!

3.2. Описание функций и особенности реализации.

С учетом изложенных выше принципов созданы две реализации ГОСТа для процессоров семейства Intel x86, близкие по быстродействию к возможному оптимуму – соответственно для 16-и и 32-х битовых процессоров. Код для 32-разрядных процессоров примерно в полтора раза быстрее соответствующего кода для 16-разрядных процессоров. Ядром является подпрограмма, реализующая универсальный базовый цикл ГОСТа. Исходные тексты всех подпрограмм приведены в качестве приложений к настоящей статье в отдельных файлах, они перечислены в следующей ниже таблице 1. Все функции являются самодокументированными, каждая описана в соответствующем файле с ее исходным текстом.

Таблица 1. Перечень файлов программной реализации ГОСТ 28147-89.

№	Функция модуля	Имя файла исх. текста	
		16 бит	32 бита
1.	Универсальный базовый цикл ГОСТа	gost.asm	gost~.asm

Таблица 1. Перечень файлов программной реализации ГОСТ 28147-89.

№	Функция модуля	Имя файла исх. текста	
		16 бит	32 бита
2.	Функция за- и расшифрования данных в режиме простой замены	simple.asm	simple~.asm
3.	Функция за- и расшифрования данных в режиме гаммирования	gamma.asm	gamma~.asm
4.	Функция зашифрования данных в режиме гаммирования с обратной связью	gammale.asm	gammale~.asm
5.	Функция расшифрования данных в режиме гаммирования с обратной связью	gammald.asm	gammald~.asm
6.	Функция вычисления имитовставки для массива данных	imito.asm	imito~.asm
7.	Функция построения расширенного ключа	expkey.asm	expkey~.asm
8.	Функция построения расширенной (1Кбайт) формы таблицы замен из обычной формы (128 байт)	expcht.asm	
9.	Функция проверки, является ли процессор, на котором исполняется приложение, 32-битовым.	—	ge386cpu.asm
10.	Заголовочный файл для использования криптографических функций в программах на языке Си	gost.h	

Комплект модулей включает функции для основных режимов шифрования, а также две вспомогательные функции, предназначенные для построения расширенных соответственно ключа и таблицы замен. Ниже изложены принципы построения программных модулей.

1. Все функции шифрования и вычисления имитовставки обрабатывают (т.е. шифруют или вычисляют имитовставку) области с размером, кратным восьми. Длина обрабатываемой области при вызове упомянутых функций задается в восьмибайтных блоках. В реальных ситуациях это не приводит к неудобству по следующим причинам:

- при шифровании простой заменой размер шифруемой области обязан быть кратным восьми байтам;
- при шифровании гаммированием (с или без обратной связи) массива данных с размером, не кратным восьми, будет также шифроваться и «мусор», содержащийся в последнем восьмибайтовом блоке за пределами значащих данных, однако его содержимое не оказывает никакого влияния на значащие данные и может не приниматься во внимание;
- при вычислении имитовставки для массивов данных их размер должен быть приведен к значению, кратному восьми, добавлением какого-либо фиксированного кода (обычно нулевых битов).

2. Криптографические функции шифрования и вычисления имитовставки позволяют выполнять обработку массивов данных по частям. Это означает, что при вызове соответствующей функции один раз для некоторой области данных и при нескольких вызовах этой же самой функции для последовательных фрагментов этой же области (естественно, их размер должен быть кратным восьми байтам, см. предыдущее замечание) будет получен один и тот же результат. Это позволяет обрабатывать данные порциями, используя буфер размером всего 8 байтов.

3. Для за- и расшифрования массива данных в режиме простой замены используется одна и та же функция. Выбор одной из двух указанных операций осуществляется заданием соответствующего расширенного ключа. Порядок следования элементов ключа должен быть взаимно обратным для указанных операций.

4. Для за- и расшифрования блока данных в режиме гаммирования используется одна и та же функция, поскольку в данном режиме зашифрование и расшифрование данных идентичны. Функция, реализующая шифрование гаммированием не осуществляет начальное преобразование синхропосылки (см. схему алгоритма на рис.5, блок 1), это необходимо выполнить с помощью явного вызова функции шифрования в режиме простой замены для синхропосылки, – это плата за возможность шифровать массив по частям.

5. Ради универсальности кода все указатели на области обрабатываемых данных сделаны дальними. Если сделать свой код для каждой модели памяти, возможно, будет достигнута некоторая ненулевая (но очень маленькая!) экономия памяти и времени выполнения, но по моему мнению, эта игра не стоит свеч.

6. Для ассемблирования (компиляции) и сборки приложенных модулей мной использовались средства разработки фирмы Borland – TASM 2.5 и выше, Borland C/C++ 2.0 и выше. При использовании других средств разработки возможно потребуется внесение изменений в исходные тексты программ.

Для иллюстрации использования представленных криптографических функций к настоящей статье приложены также текст программы шифрования файлов данных на языке Си и соответствующие файлы проекта. Эти файлы следующие:

- cryptor.c Исходные тексты программы шифрования файлов;
- gost.mak Файл проекта для 16-разрядной версии программы шифрования файлов;
- gost386.mak Файл проекта для 32-разрядной версии программы шифрования файлов.

Описание построения и синтаксиса вызова (командной строки) программы шифрования файлов также прилагается к настоящей статье.

3.3. Вопрос быстродействия.

После разработки новой программной реализации было измерено ее быстродействие, для чего был разработан комплект простых модулей, предназначенных для построения измерительной задачи. Эта задача фиксирует и выводит на дисплей время (в тактах генератора тактовой частоты таймера, 1193180 Герц), затраченное тестируемой подпрограммой на выполнение. По измеренному времени работы подпрограммы затем вычисляется (вручную) ее быстродействие как отношение количества работы ко времени ее выполнения.

Максимальная измеряемая программой длительность процесса равна $2^{32}/1193180 \approx 3599.6$ секунд, то есть примерно одному часу. Программа работает корректно и дает правильные результаты, только если запущена из ДОСа.

Для модулей ГОСТа измерялась длительность шифрования одного Мегабайта данных, которое моделировалось 32-кратным шифрованием 32-Килобайтной области памяти. Измерения проводились на машинах различных классов, результаты измерения приведены ниже в таблице 2. Для 32-битовых процессоров также приведено быстродействие 32-битовых реализаций криптографических модулей (нижнее число в соответствующей ячейке). Для сравнения также приведены измерения быстродействия реализации американского стандарта шифрования DES, опубликованной в журнале «Монитор» №7/1994. Результаты тестов показали, что быстродействие модулей для всех режимов шифрования ГОСТа примерно одинаково, а быстродействие модуля вычисления имитовставки приблизительно вдвое превышает быстродействие шифрования – что, собственно, и

ожидалось. Реализация шифрования по ГОСТ существенно (более чем в два раза) превышает исследованную реализацию DES по быстродействию.

Таблица 2. Результаты измерения быстродействия модулей шифрования

Марка компьютера, тип процессора	т.ч., МГц	Быстродействие криптографических модулей					
		gamma	gammaLD	gammaLE	simple	imito	DES
Искра 1031, К1810ВМ88	4.52	8.4	8.6	8.7	8.7	16.9	нет данных
AMI 286 Intel 80286	10	20.4	20.7	20.8	20.8	40.8	11.2
Prolinea 325 Intel 386SX-25	25	48.0 66.0	48.6 71.1	48.8 67.4	48.0 71.5	93.7 139	22.0
Неизв.модель Intel 386SX-33	33	63.8 87.6	64.5 94.5	64.7 89.5	63.8 95.0	124 185	25.9
BYTEX Intel 386DX-40	40	89 120	90 135	91 122	91 135	177 264	39.3
Acer Intel486SX33	33	114 150	113 161	114 151	114 162	226 321	41.2
Presario 460 Intel486SX2-66	66	225 298	222 319	229 303	227 324	451 637	82.2
Acer Pentium-66	66	302 351	296 397	307 355	293 405	601 777	88.7

Теперь оценим достигнутые показатели с качественной точки зрения. Предельные скорости шифрования намного превышают скорость работы платы аппаратного шифрования «Криптон–3» (до 70 Кбайт/с) и примерно соответствуют быстродействию платы «Криптон–4» (около 400 Кбайт/с). Достигнутой производительности не достаточно для действительно прозрачного шифрования данных, хранимых на жестких дисках или передаваемых через быструю сеть. Вместе с тем, быстродействия реализации вполне хватает для шифрования данных в коммутируемых каналах связи и для многих других случаев.

Можно ли еще увеличить быстродействие реализации ГОСТа? Можно, но ненамного, если оставаться в рамках формальной спецификации ГОСТа. Для этого необходимо отказаться от цикла в подпрограмме «gost», продублировав тело цикла 32 раза, как это сделал автор программного эмулятора платы «Криптон». При этом можно не разворачивать ключ в линейную последовательность элементов, но тогда для каждого базового цикла криптографического преобразования придется сделать свой программный модуль и код основного шага будет присутствовать в кодах криптографических процедур в $32+32+16=80$ экземплярах. Такой способ повышения эффективности приводит к многократному разбуханию кода при более чем скромном выигрыше в производительности, поэтому вряд ли его можно считать хорошим.

4. Вопросы использования стандарта.

4.1. Надежность реализации.

Вопрос надежности программного средства криптографической защиты это не только вопрос стойкости использованного алгоритма. Использование стойкого шифра само по себе не может сделать вашу систему надежной, хотя и является необходимым условием. Весьма важную роль играет и способ применения криптографического алгоритма. Так, в приложенной к настоящей статье программе шифрования файлов, хранение ключевой информации на дисках в открытом виде делает систему, которая была бы реализована на этой программе, потенциально нестойкой. Процедуры и правила более высокого уровня,

регламентирующие использование алгоритмов шифрования и все связанное с этим, в совокупности составляют так называемый криптографический протокол. Этот протокол определяет регламент выработки, использования, хранения и смены ключевой информации, и другие, не менее важные вопросы. Так вот, чтобы ваша система, использующая реализацию алгоритмов ГОСТа, была действительно надежна, вам необходимо будет позаботиться о разработке соответствующего протокола.

4.2. Вариации на тему ГОСТа.

Очень часто для использования в системе криптографической защиты данных требуется алгоритм с большим, чем у ГОСТа быстродействием реализации, и при этом не требуется такая же высокая как у ГОСТа криптостойкость. Типичным примером подобных задач являются различного рода биржевые торговые системы, управляющие торговыми сессиями в реальном времени. Здесь от использованных алгоритмов шифрования требуется, чтобы было невозможно расшифровать оперативные данные системы в течение сессии (данные о выставленных заявках, о заключенных сделках и т.п.), по ее истечении же эти данные, как правило, уже бесполезны для злоумышленников. Другими словами, требуется гарантированная стойкость всего на несколько часов (такова типичная продолжительность торговой сессии). Ясно, что использование полновесного ГОСТа в этой ситуации было бы стрельбой из пушки по воробьям.

К счастью, из этой ситуации есть достаточно легкий выход – использовать модификацию алгоритма ГОСТ с меньшим количеством основных шагов в базовых циклах. Этого можно достигнуть двумя путями – уменьшением длины ключа и уменьшением числа циклов использования элементов ключа – вспомните, что число основных шагов в базовых циклах шифрования равно $N = n \cdot m$, где n – число 32-битовых элементов в ключе, m – число циклов использования ключевых элементов, в стандарте $n = 8$, $m = 4$. Во сколько раз уменьшается число основных шагов в циклах, примерно во столько же раз увеличивается быстродействие кода.

К несчастью, нет никаких сведений о том, как изменяется криптостойкость подобного ослабленного варианта ГОСТа. Что касается криптоанализа по статистической линии (перебор всех возможных значений ключа), то здесь все достаточно ясно, так как эта величина определяется только размером ключа. Гораздо труднее предсказать, насколько менее сложным станет криптоанализ по алгоритмической линии (анализ уравнений преобразования данных при их шифровании).

При выборе размера «редуцированного цикла» надо принимать во внимание, что ГОСТ проектировался с учетом возможного прогресса вычислительной техники на несколько десятилетий вперед и в нем заложен огромный запас криптостойкости. По моему мнению (глубоко личному), в большинстве практических случаев представляется разумным использование редуцированных вариантов ГОСТа без изменения схемы использования ключа ($m = 4 = 3 + 1$), но с уменьшенным вчетверо размером ключа ($n = 2$), – это позволит увеличить скорость шифрования примерно вчетверо. По стойкости к статистическим методам криптоанализа данная модификация с ее 64-битовым ключом будет надежнее, чем DES с размером ключа в 56 бит.

Функции криптопреобразования, прилагающиеся к настоящей статье, допускают подобное использование, поскольку длина развернутого ключа передается в качестве параметра в каждую из подпрограмм криптографического преобразования, а подпрограмма «расширения» ключа позволяет работать с произвольной длиной ключа и схемой расширения ключа.

4.3. Необычная работа криптографической гаммы.

Конечно, основное назначение криптоалгоритмов ГОСТ – это шифрование и имитозащита данных. Однако у криптографической гаммы есть еще одно важное применение – выработка ключевой информации. Выработка массива ключевой или парольной информации большого объема является типовой задачей администратора безопасности системы. Как уже было отмечено выше, ключ может быть сгенерирован как массив нужного размера статистически независимых и равновероятно распределенных между значениями 0 и 1 битов, для этого можно использовать программу, вырабатывающую ключ по принципу «электронной рулетки». Но такой подход совершенно не годится, когда объем необходимой ключевой информации велик. В этом случае идеально использование аппаратных датчиков случайных чисел, что, однако, не всегда возможно по экономическим или техническим соображениям. В этом случае в качестве источника потока случайных битов может быть использован генератор гаммы на основе любого блочного шифра, в том числе и ГОСТ 28147-89, так как, по определению, криптографическая гамма обладает необходимыми статистическими характеристиками и криптостойкостью. Таким образом, для выработки нескольких ключей надо всего лишь сгенерировать массив данных по алгоритму выработки гаммы, и нарезать его на порции нужного размера, для стандартного варианта – 32 байта.

С паролями дело обстоит несколько сложнее. Прежде всего возникает вопрос, зачем вообще нужно их генерировать, не проще ли по мере надобности брать их из головы. Несостоятельность такого подхода была наглядно продемонстрирована серией инцидентов в компьютерных сетях, самым крупным из которых был суточный паралич сети Internet в ноябре 1988 года. Одним из способов доступа злоумышленной программы в систему был подбор паролей: программа пыталась войти в систему, последовательно перебирая пароли из своего внутреннего списка в несколько сотен, причем в значительной доле случаев ей это удавалось сделать, – фантазия человека по выдумыванию паролей оказалась очень бедной. Именно поэтому в тех организациях, где безопасности уделяется должное внимание, пароли генерирует и раздает пользователям системный администратор по безопасности.

Выработка паролей чуть сложнее, чем выработка ключей, так как при этом «сырую» двоичную гамму необходимо преобразовать к символьному виду, а не просто «нарезать» на куски. Основное, на что необходимо обратить внимание при этом – обеспечение равной вероятности появления каждого из символов алфавита.

Исходные тексты программ выработки массива паролей и ключей на языке Си приложены к настоящей статье.