МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Некоммерческое акционерное общество

АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ

ФАКУЛЬТЕТ «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

Допущен к защите:

зав. кафедрой «Компьютерные технологии»

д. ф.-м. н., профессор _____ Куралбаев З.К.

«___»_____20 __г.

Магистерская диссертация

Влияние видов алгоритмов шифрования на качество информационных систем

специальность: 6М070400 - Вычислительная техника и программное обеспечение

Магистрант Касим Оспандияр Данияруулы

_____ Касим О.Д.

Научный руководитель,

к. т. н., ст. преподаватель _____ Сербин В.В.

АЛМАТЫ 2014

ОГЛАВЛЕНИЕ

Введение ... 3
1. Обзор и анализ современных алгоритмов шифрования ... 7
1.1 Алгоритм шифрования AES ... 7
1.2 Алгоритм шифрования Serpent ... 16
1.3 Алгоритм шифрования Twofish ... 24
1.4 Алгоритм шифрования MARS ... 33
1.5 Алгоритм шифрования RC6 ... 44
1.6. Описание показателей алгоритмов шифрования ... 47
1.7 Влияние алгоритмов шифрования на качество информационных систем ... 48
2.Парольная система авторизации с применением технологии шифрования методом “нелинейной динамики” Captcha ... 52
2.1 Аутентификация как базис безопасности ... 52
2.2 Парольная аутентификация ... 53
2.3 Противодействие подбору паролей ... 55
2.4 Новый метод создания паролей с помощью CAPTCHA ... 56
2.5 Модифицированный алгоритм шифрование двухфакторной аутентификации с применением Captcha ... 56
3.Исследование влияния алгоритмов шифрования на качество информационных систем ... 62
3.1 Криптоанализ MD5 ... 64
3.2 Описание экспериментальной модели ... 65
3.3 Экспериментальная часть ... 66
3.4 Анализ результатов эксперимента по уязвимости ... 67
3.5 Анализ результатов эксперимента по эффективности от криптоаналитических атак ... 67
3.6 Анализ результатов эксперимента по эффективности различных видов атак от типа пароля ... 68
3.7 Анализ результатов эксперимента эффективности по времени ... 69
3.8 Анализ результатов эксперимента надежности ... 70
Приложение А ... 72
Приложение Б ... 73
Приложение В ... 74
Приложение Г ... 75
Приложение Д ... 76
Приложение Е ... 77
Заключение ... 78
Список используемой литературы ... 80

АҢДАТПА

Диссертациялық жұмыс қазіргі алгоритмның крифтографиялық қажырлылығының байқауына арнаулы, ақпараттық жүйенің пайдаланушысының айтылмыш есептік жазбасының шифрлауы үшін. Айтылмыш жұмыстың мақсаты MD5 хеш-функция алгоритмының крифтографиялық қажырлылығының зерттеуі болып табылады. Байқау мүмкіндік игерушілік модифицирілінген алгоритм шифрлау сенімділіктің көтермелеуі үшін ақпараттық жүйе. Құрулы мақсаттың табысы үшін жұмыста криптоаналитикалық үдерістің математикалық модельдеуінің әдістері пайдаланылды, шифрлаудың және мезгілдіктің кідірісінің ұйғарымінің үдерісінің компьютерлік модельдеуінің әдісінің математикалық модельдеуінің әдістері, шифрлаудың алгоритмының крифтографиялық қажырлылығының қағидасы.

АННОТАЦИЯ

Диссертационная работа посвящена изучению криптографической стойкости современных алгоритмов, применяемых для шифрования данных учетных записей пользователей информационных систем.
Целью данной работы является исследование криптографической стойкости алгоритмов основанных на безе хеш-функции MD5. Изучение возможности использования модифицированных алгоритмов шифрования для повышения надежности информационных систем. Для достижения поставленных целей в работе использовались методы математического моделирования криптоаналитических процессов, метода компьютерного моделирования процессов шифрования и определения временной задержки, теория криптографической стойкости алгоритмов шифрования.

ABSTRACT

Dissertation work is devoted to studying of cryptographic firmness of the modern algorithms applied to enciphering of these accounts of users of information systems.
The purpose of this work is research of cryptographic firmness of algorithms based on meringue MD5 hash function. Studying of possibility of use of the modified

algorithms of enciphering for increase of reliability of information systems. For achievement of goals in work methods of mathematical modeling of cryptoanalytical processes, a method of computer modeling of processes of enciphering and definition of a temporary delay, the theory of cryptographic firmness of algorithms of enciphering were used.

Введение

В настоящее время основным способом защиты информации от несанкционированного доступа (НСД) является внедрение так называемых средств А А А (Authentication, Authorization, Accounting — аутентификация, авторизация, управление правами пользователей). При использовании этой технологии пользователь получает доступ к компьютеру лишь после того, как успешно прошел процедуры идентификации и аутентификации. Стоит учесть , что на мировом рынке ИТ-услуг сегмент А А А постоянно растет. Эта тенденция подчеркивается в аналитических обзорах IDC, Gartner и других консалтинговых фирм. Такой же вывод можно сделать , внимательно просмотрев ежегодный обзор компьютерной преступности на рисунке 1, Института компьютерной безопасности США и ФБР за 2005 год. Суммарный объем потерь за 2005 год — 130 104 542 долл. Количество предприятий-респондентов (США) — 700

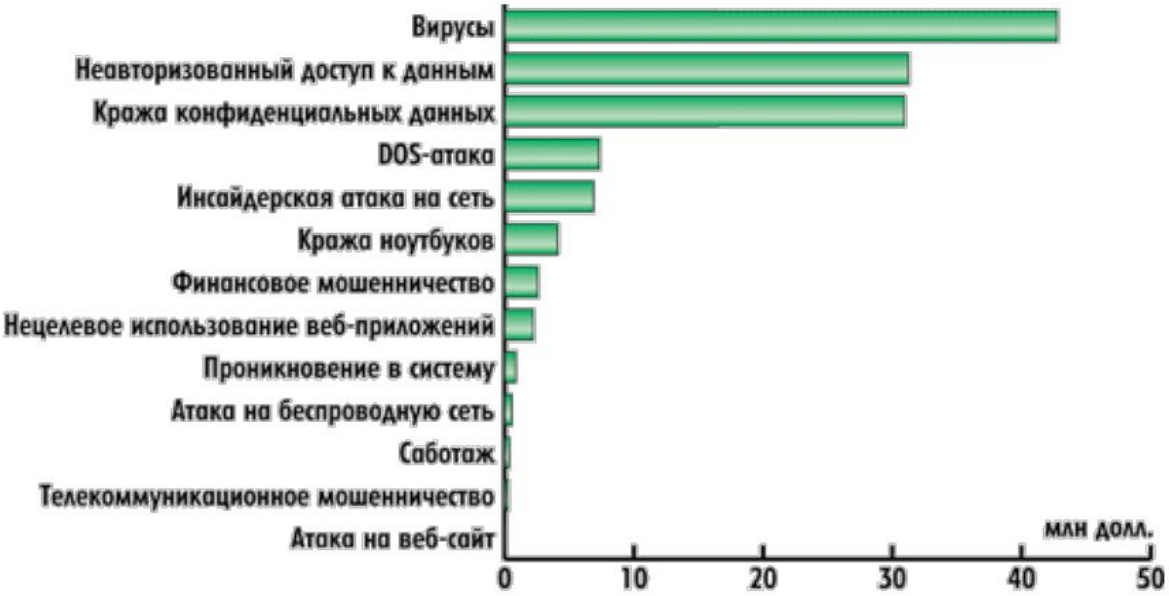


Рисунок 1 - Данные по объему потерь от разных видов атак за 2005 год, долл.

Как видно из диаграммы, ущерб от краж и конфиденциальной информации значительно увеличился. То есть каждая из опрошенных компаний потеряла

в среднем более 350 тыс. долл. вследствие краж и конфиденциальной информации. Это исследование подтверждает тенденции, наметившиеся в последние несколько лет. Согласно отчету Института компьютерной безопасности США и ФБР за 2004 год, кража чувствительных данных уже тогда входила в число опаснейших угроз — ущерб от нее составлял около 40% от общего объема ущерба всех его угроз. При этом средний объем потерь был равен более 300 тыс. долл. , а максимальный объем — 1,5 млн долл. Исходя из этого, можно сделать вывод, что кража конфиденциальной информации имеет один из наиболее
высок их рейтингов среди всех ИТ-у гроз в США . Стоит отметить , что найти виновного без решения вопросов идентификации и аутентификации невозможно!

Отметим, что вопросы разграничения доступа решаются в обязательном порядке при создании любой информационной системы. В наше время, когда системы становятся все более распределенными, трудно переоценить важность корректного разграничения доступа. При этом требуется все более надежная защита систем аутентификации как от внешних, так и от внутренних злоумышленников. Стоит понимать, что пользователи не склонны усложнять себе жизнь и стараются пользоваться как можно менее сложными паролями. А следовательно, для устранения этого в дальнейшем все чаще будут применяться программно-аппаратные средства аутентификации, которые постепенно придут на смену традиционным паролям, что видно на рисунке 2.

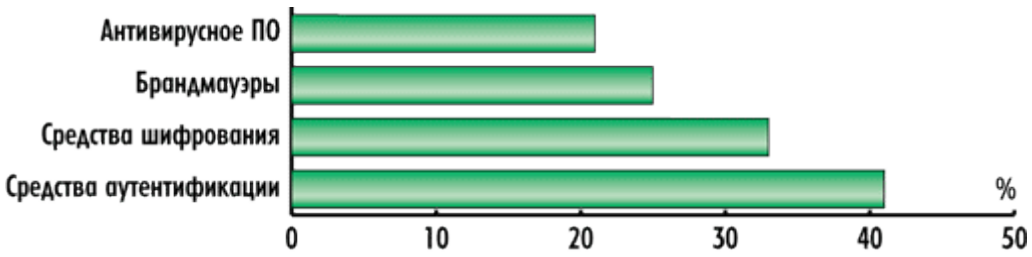


Рисунок 2 - Рост рынка средств информационной безопасности

Актуальность темы обусловлена необходимостью поиска новых способов шифрования, не требующих больших временных затрат и являющихся наиболее криптостойкими. На сегодняшний день применение различных алгоритмов решает актуальную техническую задачу, связанную с идентификацией информации и аутентификацией пользователей в системах. Наглядным примером таких систем являются популярные социальные сети, такие как «ВКонтакте», «Одноклассники», «Facebook» и т. д., а так же почтовые сервисы Mail.RU, Hotmail.com, gmail.com и т. д., веб-форумы, пиринговые сети (P2P, eMule, eDonkey, eD2k и т. д.). Почти все подобные системы используют для защиты конфиденциальных данных алгоритмы без дополнительного шифрования конечной суммы (хеш-сумма). Ежедневно

злоумышленники, используя похищенные аутентификационные данные, наносят огромный вред тысячам пользователям сети Интернет. По этой причине вопрос о необходимости дополнительного шифрования стоит наиболее актуальным для качественной защиты различных информационных систем, в том числе, таких как интернет банкинг, интернет магазин и т.д.

Цель проекта. Исследование криптографической стойкости различных алгоритмов шифрования; Изучение возможности использования модифицированных алгоритмов шифрования для повышения надежности информационных систем.

Задачи исследования. Изучить особенности различных алгоритмов, рассмотреть возможности повышений криптостойкости, охарактеризовать различные типы существующих методов криптографического анализа;
Создать модель криптоаналитической атаки на серверы, хранящие конфиденциальную информациею, с возможность ввода различных предустановочных параметров;
Используя созданную модель, провести исследование криптографической стойкости алгоритмов, используемых в современных сетях связи;
Сделать выводы о возможности повышения надежности шифрования;

Методы исследования. Экспериментальный метод исследования работы алгоритма основан на виртуальной модели криптоаналитической атаки на сервера, хранящих данные учетных записей с различными исходными параметрами.

Объект исследования. Защита информации: криптография, криптоанализ.

Предмет исследования. Криптографические алгоритмы: хеш-функция MD5, алгоритмы шифрования AES,Serpent,Twofish,MARS,RC6,Base64, Unix, HMAC. Методы криптографического анализа: полный перебор (Brute Force), атака с использованием словаря (Dictionary attack), атака с использованием радужных таблиц (Rainbow attack).

Гипотеза. Используемые на сегодня методы шифрования данных учетных записей на серверах не обеспечивают надежность на необходимом уровне. Если использовать дополнительное шифрование конечных сумм данных, то можно добиться высокого уровня защищенности при незначительных затратах времени и мощности, используемых на вторичное шифрование данных.

Новизна или личный вклад. На данный момент исследования в сфере шифрования аутентификационных данных фактически направлены на создание новых алгоритмов или протоколов передачи информации. Фактически исследования возможности использования вторичного шифрования не ведутся. Модель, которую можно было бы использовать в исследовании криптостойкости алгоритмов шифрования, не существует. Данная работа доказывает целесообразность использования дополнительного шифрования хеш-сумм с помощью captcha для значительного увеличения

защищенности информации. Созданная модель позволяет продолжить исследования в данной сфере.

Практическая значимость. Исследование позволяет выявить слабые стороны используемой на сегодня технологии хеширования, демонстрирует возможности вторичного шифрования, позволяющего значительно повысить криптографическую стойкость зашифрованной информации, обеспечить защищенность учетных записей пользователей в различных информационных системах.

1. Обзор и анализ современных алгоритмов шифрования

1.1 Алгоритм шифрования AES

Advanced Encryption Standard (AES), также известный как Rijndael — симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES. Этот алгоритм хорошо проанализирован и сейчас широко используется, как это было с его предшественником DES. Национальный институт стандартов и технологий США опубликовал спецификацию AES 26 ноября 2001 года после пятилетнего периода, в ходе которого были созданы и оценены 15 кандидатур. 26 мая 2002 года AES был объявлен стандартом шифрования. По состоянию на 2009 год AES является одним из самых распространённых алгоритмов симметричного шифрования.

Для AES длина input(блока входных данных) и State(состояния) постоянна и равна 128 бит, а длина шифроключа **K** составляет 128, 192, или 256 бит. При этом, исходный алгоритм Rijndael допускает длину ключа и размер блока от 128 до 256 бит с шагом в 32 бита. Для обозначения выбранных длин input, State и Cipher Key в 32-битных словах используется нотация Nb = 4 для input и State, Nk = 4, 6, 8 для Cipher Key соответственно для разных длин ключей.

В начале шифрования input копируется в массив State по правилу:

$$\mathrm{state}[r, c] = \mathrm{input}[r + 4c],$$

для $0 \leq r < 4$ и $0 \leq c < Nb$.

После этого к State применяется процедура AddRoundKey() и затем State проходит через процедуру трансформации (раунд) 10, 12, или 14 раз (в зависимости от длины ключа), при этом надо учесть, что последний раунд несколько отличается от предыдущих. В итоге, после завершения последнего раунда трансформации, State копируется в output по правилу:

$$\mathrm{output}[r + 4c] = \mathrm{state}[r, c],$$

для $0 \leq r < 4$ и $0 \leq c < Nb$.

Отдельные трансформации SubBytes(), ShiftRows(), MixColumns(), и AddRoundKey() — обрабатывают State. Массив w[] — содержит key schedule.

1.1.2 Функция зашифрования

Введем следующие обозначения:

SubBytes() - замена байтов- побайтовая нелинейная подстановка в State-блоках (S-Box) с использованием фиксированной таблицы замен размерностью 8x256 (affain map table);

ShiftRows() - сдвиг строк - циклический сдвиг строк массива State на различное количество байт;

MixColumns() - перемешивание столбцов - умножение столбцов состояния, рассматриваемых как многочлены над GF(28);

AddRoundKey() - сложение с раундовым ключом - поразрядное XOR содержимого State с текущим фрагментом развернутого ключа.

На псевдокоде операция зашифрования выглядит следующим образом:

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]
    state = in
  AddRoundKey(state, w[0, Nb-1])
  for round = 1 step 1 to Nr-1
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
  end for
  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  out = state
end
```

После заполнения массива State элементами входных данных к нему применяется преобразование AddRoundKeyQ, далее, в зависимости от величины Nk, массив State подвергается трансформации раундовой 10, 12 или 14 раз, причем в финальный раунд является несколько укороченным - в нем отсутствует преобразование MixColumnsQ. Выходными данными описанной последовательности операций является шифротекст - результат действия функции зашифрования AES.

1.1.3 Функции расшифрования

В спецификации алгоритма AES предлагаются два вида реализаций функции расшифрования отличающихся друг от друга последовательностью приложения преобразований обратных преобразованиям функции зашифрования и последовательностью планирования ключей (см. ниже).

Введем следующие обозначения:

InvSubBytes() — обратная SubBytes() замена байтов- побайтовая нелинейная подстановка в SWe-блоках с использованием фиксированной таблицы замен размерностью 8x256 (inverse affain map);

InvShiftRows() — обратный сдвиг строк ShiftRows()— циклический сдвиг строк массива State на различное количество байт;

InvMixColumns() - восстановление значений столбцов - умножение столбцов состояния, рассматриваемых как многочлены над GF(28);

Функция обратного расшифрования

Если вместо SubBytes{), ShiftRows{), MixColumns{) и AddRoundKey{) в обратной последовательности выполнить инверсные им преобразования, можно построить функцию обратного расшифрования. При этом порядок использования раундовых ключей является обратным по отношению к тому, который используется при зашифровании показанный на псевдокоде:

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte state[4,Nb]
      state = in
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)
   end for
   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])
   out = state
end
```

Алгоритм обратного расшифрования, описанный выше имеет порядок приложения операций-функций обратный порядку операций в алгоритме прямого зашифрования, но использует те же параметры развёрнутого ключа.

Два следующих свойства позволяют сделать это:

- Порядок приложения функций SubBytes() и ShiftRows() не играет роли. То же са мое верно и для операций InvSubBytes() и InvShiftRows(). Это происходит потому, что функции SubBytes() и InvSubBytes() работают с байтами , а операции ShiftRows() и InvShiftRows() сдвигают целые байты, не затрагивая их значений.
- Операция MixColumns() является линейной относительно входных данных, что означает InvMixColumns(State XOR RoundKey) = = InvMixColumns(State) XOR InvMixColumns(RoundKey)

Эти свойства функций алгоритма шифрования позволяют изменить порядок применения функций InvSubBytes() и InvShiftRows(). Функции AddRounKey() и InvMixCol-umns() также могут быть применены в обратном порядке, но при условии, что столбцы (32-битные слова) развёрнутого ключа расшифрования предварительно пропущены через функцию InvMixColumns().

Таким образом, можно реализовать более эффективный способ расшифрования с тем же порядком приложения функций как и в алгоритме зашифрования.

Алгоритм выработки ключей (Key Schedule)

Введем следующие обозначения: Rconf - массив 32-битных раундовых констант;

RotWord() — операция циклической перестановки входного 4-байтного слова в выходное по следующему правилу [a0, ai, a2, a3] —> [ax, a2, a3, a0];

SubWord() - операция замены в 4-байтном слове с помощью S-Box каждого байта;

⊕ - операция исключающего или XOR. Операция планирования (расширения) ключа реализованная на псевдокоде:

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word temp
   i = 0;
      while ( i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while
      i = Nk
   while ( i < Nb * (Nr+1))
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
```

```
      i = i + 1
   end while
end
```

Раундовые ключи получаются из ключа шифрования посредством алгоритма выработки ключей. Он содержит два компонента: расширение ключа (Key Expansion) и выбор раундового ключа (Round Key Selection). Основополагающие принципы алгоритма выглядят следующим образом:

общее число битов раундовых ключей равно длине блока, умноженной на число раундов, плюс 1;

ключ шифрования расширяется в расширенный ключ (Expanded Key);

раундовые ключи берутся из расширенного ключа следующим образом: первый раундовый ключ содержит первые Nb слов, второй - следующие Nb слов и т. д.

Расширение (планирование) ключа

Расширенный ключ показан на рисунке 1.1, представляет собой линейный массив w[i] состоящий из A(Nr +1) 4-байтовых слов, i = O,4(Nr +1).

Рисунок 1.1 - Процедуры расширения и выборки раундового ключа для Nk = 4

Светло-серым цветом выделены слова расширенного ключа, которые формируются без использования функций SubWord() и RotWord().

Темно-серым цветом ,выделены слова расширенного ключа, при вычислении которых используются преобразования SubWord() и RotWord())»

Первые Nk слов содержат ключ шифрования. Каждое последующее слово w[i] получается посредством XOR предыдущего слова w[i-1] и слова на Nk позиций ранее:

w[i- Nk]: w[i]= w[i-1] ⊕ w[i- Nk].

Для слов, позиция которых кратна Nk, перед XOR применяется преобразование к w[i-1], а затем еще прибавляется раундовая константа Rcon[i] . Преобразование реализуется с помощью двух дополнительных функций: RotWord() и SubWord().

Значение Rcon[j] равно 2j-1 . Значение w[i] в этом случае определяется выражением: w[i] = SubWord(RotWord(w[i-1]) ⊕ Rcon[i/Nk] ⊕ M[i-Nk].

Выбор раундового ключа i-тый раундовый ключ выбирается из слов массива расширенного ключа в промежутке от W[Nb * i] до W[Nb * (i+1)].

Для функции зашифрования расширенный ключ генерируется алгоритмом как показано на рисунке 1.1. Для функции обратного расшифрования используется этот же ключ, но в обратной последовательности начиная с последнего раундового подключа зашифрования.

Для функции прямого расшифрования используется несколько модифицированный алгоритм планирования ключа. При формировании развёрнутого ключа в процедуру планирования необходимо добавить в конце дополнительное преобразование, как показано на рисунке 1.2 причем расширенный ключ используется при прямом расшифровании в той же последовательности, что и при зашифровании.

```
for i = 0 step 1 to (Nr+1)*Nb-1
   dw[i] = w[i]
end for

for round = 1 step 1 to Nr-1
   InvMixColumns(dw[round*Nb, (round+1)*Nb-1])
end for
```

Рисунок 1.2 - Дополнительное преобразование расширенного ключа для функции прямого расшифрования

1.1.4 Раундовое преобразование

Раундовое преобразование состоит из последовательного применения к массиву State ряда трансформаций.. Сейчас обсудим детали его реализации.

Нелинейная замена байтов массива состояния посредством трансформации SubBytesQ имеет вид:

$$\lambda(f) \equiv ((X^4 + X^3 + X^2 + X + 1) \cdot f + X^6 + X^5 + X + 1) \bmod (X^8 + 1)$$

Многократное вычисление в процессе зашифрования данного выражения оказывало бы неоправданную вычислительную нагрузку на исполняющую систему, поэтому для практической реализации наиболее приемлемым решением является использование предварительно вычисленной таблицы замены S-Box. Логика работы S-Box при преобразовании байта {xy} отражена в шестнадцатеричном виде на рисунке 1.3:

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Рисунок 1.3 - Таблица S-Box замены байт

Ее использование сводит операцию SubBytesQ к простейшей выборке байта из массива λ(f) = Sbox[f].

В функциях расшифрования применяется операция обратная InvSub-Bytes().

Она реализуется так же просто, как и предыдущая посредством инверсной таблицы S-Box – λ-1(f) = InvSbox[f]., ее логика работы при преобразовании байта {xy} отражена в шестнадцатеричном виде на рисунке 1.4

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Рисунок 1.4 - Таблица S-Box инверсной замены байт

Рисунок 1.5 иллюстрирует применение преобразования замены байт к состоянию в функциях зашифрования и расшифрования:

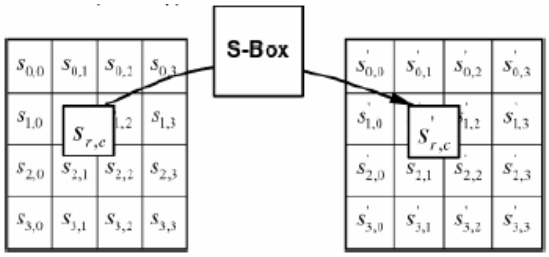


Рисунок 1.5 - Преобразование состояния посредством таблицы замены S-Box

В преобразовании сдвига строк показанных на рисунке 1.6 (ShiftRows) последние 3 строки состояния циклически сдвигаются ВЛЕВО на различное число байтов. Строка 1 сдвигается на C1 байт, строка 2 - на C2 байт, и строка 3 - на Cз байт. Значения сдвигов C1, C2 и C3 в Rijndael зависят от длины блока Nb .

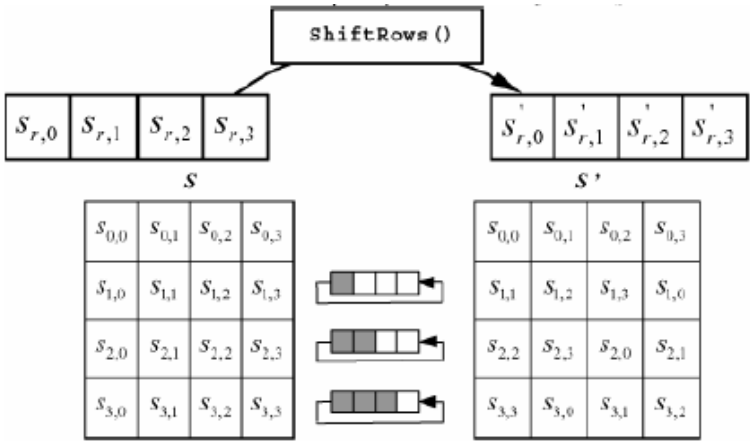


Рисунок 1.6 - Преобразование сдвига строк в функции зашифрования

В преобразовании обратного сдвига строк InvShiftRows последние 3 строки состояния циклически сдвигаются ВПРАВО на различное число байтов. Строка 1 сдвигается на C1байт, строка 2 - на C2 байт, и строка 3 - на C3 байт.

Перемешивание столбцов

В преобразовании перемешивания столбцов (MixColumns) столбцы состояния рассматриваются как многочлены над GF(2S) и подвергаются преобразованию /j,(g) = c * gmod(Y4 +1), где c = (X,1,1,X +1), т.е умножаются по модулю x4 + 1 на многочлен c(x), выглядящий, как: c(x) = {03}x3 + {01}x2 + {01}x + {02}.

Это преобразование может быть представлено в матричном виде следующим образом:

$$\begin{bmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{bmatrix}, \quad 0 \le c \le 3, \text{ где } c - \text{номер столбца массива } State.$$

Применение этой операции ко всем четырем столбцам состояния обозначается, как MixColumns(State). Рисунок 1.7 демонстрирует применение преобразования MixColumnsQ к столбцу состояния.

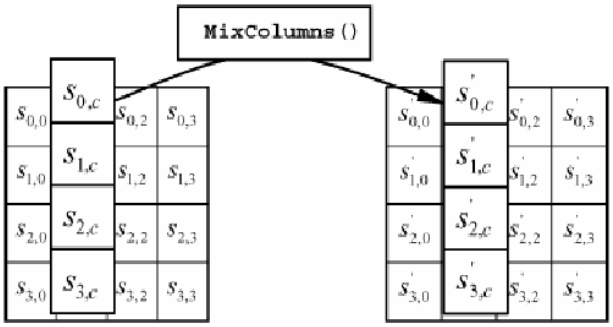


Рисунок 1.7 - Операция перемешивания действует на столбцы состояния

В обратном преобразовании InvMixColumnsQ столбцы состояния рассматриваются как многочлены над GF(2S), но, естественно, подвергаются обратному преобразованию v(g) = ^-1(g) = d-gmod(Y4+l), где d = (X3+X2 + X,X3+l,X3+X2+l,X3+X + 1), т.е умножаются по модулю x4 + 1 на многочлен d(x), выглядящий следующим образом: d(x) = {Ob}x3 + {0d}x2 + {09}x + {0e}.

Это может быть представлено в матричном виде следующим образом:

$$\begin{bmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{bmatrix}, \quad 0 \le c \le 3 \text{ , где } c - \text{номер столбца массива } State..$$

Добавление раундового ключа AddRoundKey()

В данной операции раундовый ключ добавляется к состоянию посредством поразрядного XOR. Длина ключа (в 32-разрядных словах) равна длине блока Nb .

Рисунок 1.8 иллюстрирует действие данной операции на состояние. Это преобразование обозначается как AddRoundKey(State, RoundKey).

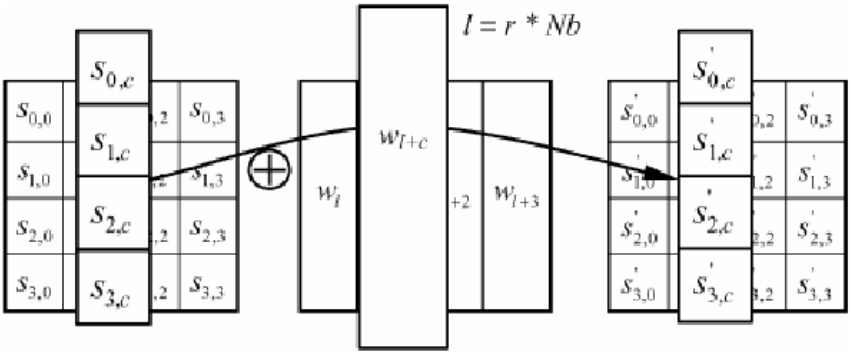


Рисунок 1.8 - Добавление раундового ключа AddRoundKey()

Основные особенности AES

В заключении сформулируем основные особенности AES:

- новая архитектура «Квадрат», обеспечивающая быстрое рассеивание и перемеши вание информации, при этом за один раунд преобразованию подвергается весь входной блок;
- байт-ориентированная структура, удобная для реализация на 8-разрядных микро контроллерах;
- все раундовые преобразования суть операции в конечных полях, допускающие эффективную аппаратную и программную реализацию на различных платформах.

1.2 Алгоритм шифрования Serpent

Алгоритм Serpent разработан тремя известнейшими криптологами: Россом Андерсоном, Эли Бихамом и Ларсом Кнудсеном. Каждый из них знаменит своими криптоаналитическими работами, а также разработанными ранее алгоритмами шифрования (например, широкую известность получили алгоритмы Bear и Leon, разработанные Андерсоном и Бихамом; не менее известен алгоритм Square, разработанный авторами алгоритма Rijndael Джоан Деймен (Joan Daemen) и Винсентом Риджменом (Vincent Rijmen) при участии Ларса Кнудсена). Именно Эли Бихама без преувеличения можно назвать величайшим криптоаналитиком современности — его авторству (часто в соавторстве с другими специалистами) принадлежит множество работ, посвященных методам вскрытия различных известных алгоритмов шифрования.

Еще до конкурса AES появился алгоритм Serpent-0, отличающийся от присланного на конкурс алгоритма Serpent-1 (или просто Serpent) только тем, что в нем были использованы таблицы замен алгоритма DES (в незначительно модифицированном виде). В Serpent используются уже оригинальные таблицы замен, которые, по словам авторов, вкупе с незначительным изменением процедуры расширения ключа усилили алгоритм против дифференциального и линейного криптоанализа.

Рассмотрим здесь именно тот алгоритм Serpent, который был прислан на конкурс AES и стал одним из его финалистов.

1.2.1 Структура алгоритма

Serpent представляет собой SP-сеть, в которой блок данных в процессе шифрования разбивается на 4 субблока по 32 бита проиллюстрировано на рисунке 1.9.

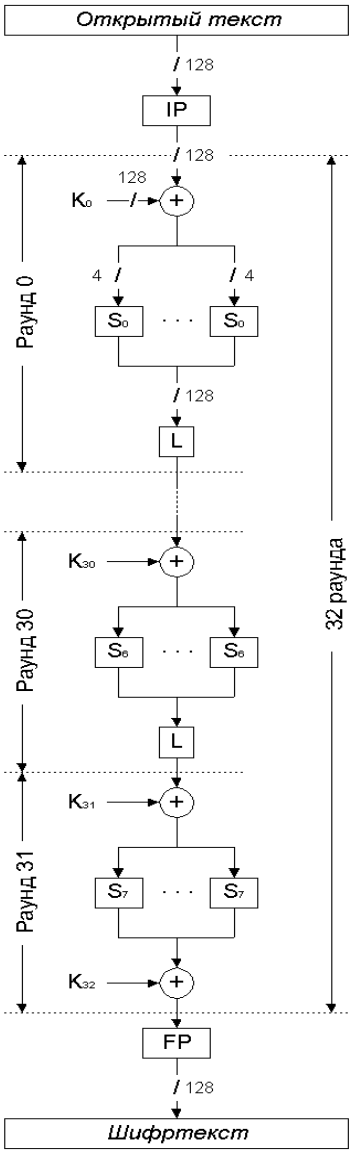


Рисунок 1.9 - Структура алгоритма Serpent.

Алгоритм выполняет 32 раунда преобразований; перед первым раундом выполняется начальная перестановка IP, после заключительного раунда — финальная перестановка FP. Начальная перестановка определена следующим образом:

0	32	64	96	1	33	65	97	2	34	66	98	3	35	67	99

4	36	68	100	5	37	69	101	6	38	70	102	7	39	71	103
8	40	72	104	9	41	73	105	10	42	74	106	11	43	75	107
12	44	76	108	13	45	77	109	14	46	78	110	15	47	79	111
16	48	80	112	17	49	81	113	18	50	82	114	19	51	83	115
20	52	84	116	21	53	85	117	22	54	86	118	23	55	87	119
24	56	88	120	25	57	89	121	26	58	90	122	27	59	91	123
28	60	92	124	29	61	93	125	30	62	94	126	31	63	95	127

Это означает, что бит 0 остается на своем месте, бит 32 входного значения становится битом 1, бит 64 — битом 2 и т.д.
Финальная перестановка является инверсной начальной перестановке и выглядит так:

0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
64	68	72	76	80	84	88	92	96	100	104	108	112	116	120	124
1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
65	69	73	77	81	85	89	93	97	101	105	109	113	117	121	125
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
66	70	74	78	82	86	90	94	98	102	106	110	114	118	122	126
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63
67	71	75	79	83	87	91	95	99	103	107	111	115	119	123	127

Функция раунда весьма проста и состоит из следующих операций:

1. Наложение 128-битного ключа раунда побитовой логической операцией «исключающее или» (XOR).
2. Табличная замена. Обрабатываемый 128-битный блок данных разбивается на 32 фрагмента по 4 бита, над каждым из которых выполняется табличная замена 4 * 4 бит.

Полученный результат объединяется обратно в 128-битный блок.
В каждом раунде используется одна и та же таблица; всего же в алгоритме определены 8 таблиц замен S_0...S_7, которые циклически используются в 32 раундах шифрования:

S_0	3	8	15	1	10	6	5	11	14	13	4	2	7	0	9	12
S_1	15	12	2	7	9	0	5	10	1	11	14	8	6	13	3	4
S_2	8	6	7	9	3	12	10	15	13	1	14	4	0	11	5	2
S_3	0	15	11	8	12	9	6	3	13	1	2	4	10	7	5	14
S_4	1	15	8	3	12	0	11	6	2	5	4	10	9	14	7	13
S_5	15	5	2	11	4	10	9	12	0	3	14	8	13	6	7	1
S_6	7	2	12	5	8	4	6	11	14	9	1	15	13	3	10	0

S_7	1	13	15	0	14	8	2	11	7	4	12	10	9	3	5	6

Ячейки таблицы содержат выходные значения; входное значение определяет требуемый номер столбца. Например, таблица S_0 меняет 0 на 3, 1 на 8 и т.д. Таблицы замен алгоритма Serpent определенным образом сгенерированы из таблиц алгоритма DES. В спецификации алгоритма приведен псевдокод, использованный для генерации таблиц S_0...S_7.

Линейное преобразование блока данных, которое определяется следующей таблицей:

16, 52, 56, 70, 83, 94, 105	72, 114, 125	2, 9, 15, 30, 76, 84, 126	36, 90, 103
20, 56, 60, 74, 87, 98, 109	1, 76, 118	2, 6, 13, 19, 34, 80, 88	40, 94, 107
24, 60, 64, 78, 91, 102, 113	5, 80, 122	6, 10, 17, 23, 38, 84, 92	44, 98, 111
28, 64, 68, 82, 95, 106, 117	9, 84, 126	10, 14, 21, 27, 42, 88, 96	48, 102, 115
32, 68, 72, 86, 99, 110, 121	2, 13, 88	14, 18, 25, 31, 46, 92, 100	52, 106, 119
36, 72, 76, 90, 103, 114, 125	6, 17, 92	18, 22, 29, 35, 50, 96, 104	56, 110, 123
1, 40, 76, 80, 94, 107, 118	10, 21, 96	22, 26, 33, 39, 54, 100, 108	60, 114, 127
5, 44, 80, 84, 98, 111, 122	14, 25, 100	26, 30, 37, 43, 58, 104, 112	3, 118
9, 48, 84, 88, 102, 115, 126	18, 29, 104	30, 34, 41, 47, 62, 108, 116	7, 122
2, 13, 52, 88, 92, 106, 119	22, 33, 108	34, 38, 45, 51, 66, 112, 120	11, 126
6, 17, 56, 92, 96, 110, 123	26, 37, 112	38, 42, 49, 55, 70, 116, 124	2, 15, 76
10, 21, 60, 96, 100, 114, 127	30, 41, 116	0, 42, 46, 53, 59, 74, 120	6, 19, 80
3, 14, 25, 100, 104, 118	34, 45, 120	4, 46, 50, 57, 63, 78, 124	10, 23, 84
7, 18, 29, 104, 108, 122	38, 49, 124	0, 8, 50, 54, 61, 67, 82	14, 27, 88
11, 22, 33, 108, 112, 126	0, 42, 53	4, 12, 54, 58, 65, 71, 86	18, 31, 92
2, 15, 26, 37, 76, 112, 116	4, 46, 57	8, 16, 58, 62, 69, 75, 90	22, 35, 96
6, 19, 30, 41, 80, 116, 120	8, 50, 61	12, 20, 62, 66, 73, 79, 94	26, 39, 100
10, 23, 34, 45, 84, 120, 124	12, 54, 65	16, 24, 66, 70, 77, 83, 98	30, 43, 104
0, 14, 27, 38, 49, 88, 124	16, 58, 69	20, 28, 70, 74, 81, 87, 102	34, 47, 108
0, 4, 18, 31, 42, 53, 92	20, 62, 73	24, 32, 74, 78, 85, 91, 106	38, 51, 112
4, 8, 22, 35, 46, 57, 96	24, 66, 77	28, 36, 78, 82, 89, 95, 110	42, 55, 116
8, 12, 26, 39, 50, 61, 100	28, 70, 81	32, 40, 82, 86, 93, 99, 114	46, 59, 120
12, 16, 30, 43, 54, 65, 104	32, 74, 85	36, 90, 103, 118	50, 63, 124
16, 20, 34, 47, 58, 69, 108	36, 78, 89	40, 94, 107, 122	0, 54, 67
20, 24, 38, 51, 62, 73, 112	40, 82, 93	44, 98, 111, 126	4, 58, 71
24, 28, 42, 55, 66, 77, 116	44, 86, 97	2, 48, 102, 115	8, 62, 75
28, 32, 46, 59, 70, 81, 120	48, 90, 101	6, 52, 106, 119	12, 66, 79
32, 36, 50, 63, 74, 85, 124	52, 94, 105	10, 56, 110, 123	16, 70, 83
0, 36, 40, 54, 67, 78, 89	56, 98, 109	14, 60, 114, 127	20, 74, 87

4, 40, 44, 58, 71, 82, 93	60, 102, 113	3, 18, 72, 114, 118, 125	24, 78, 91
8, 44, 48, 62, 75, 86, 97	64, 106, 117	1, 7, 22, 76, 118, 122	28, 82, 95
12, 48, 52, 66, 79, 90, 101	68, 110, 121	5, 11, 26, 80, 122, 126	32, 86, 99

Каждая ячейка таблицы соответствует биту выходного значения (от 0 до 127); в ячейке перечислены входные биты, XOR которых дает выходное значение. Например:

$$Y_0 = X_{16} \text{ A } X_{52} \text{ A } X_{56} \text{ A } X_{70} \text{ A } X_{83} \text{ A } X_{94} \text{ A } X_{105},$$

$$Y_1 = X_{72} \text{ A } X_{114} \text{ A } X_{125} \text{ и т.д.},$$

где X_n и Y_n — соответственно, **n**-й бит входного и выходного значения.

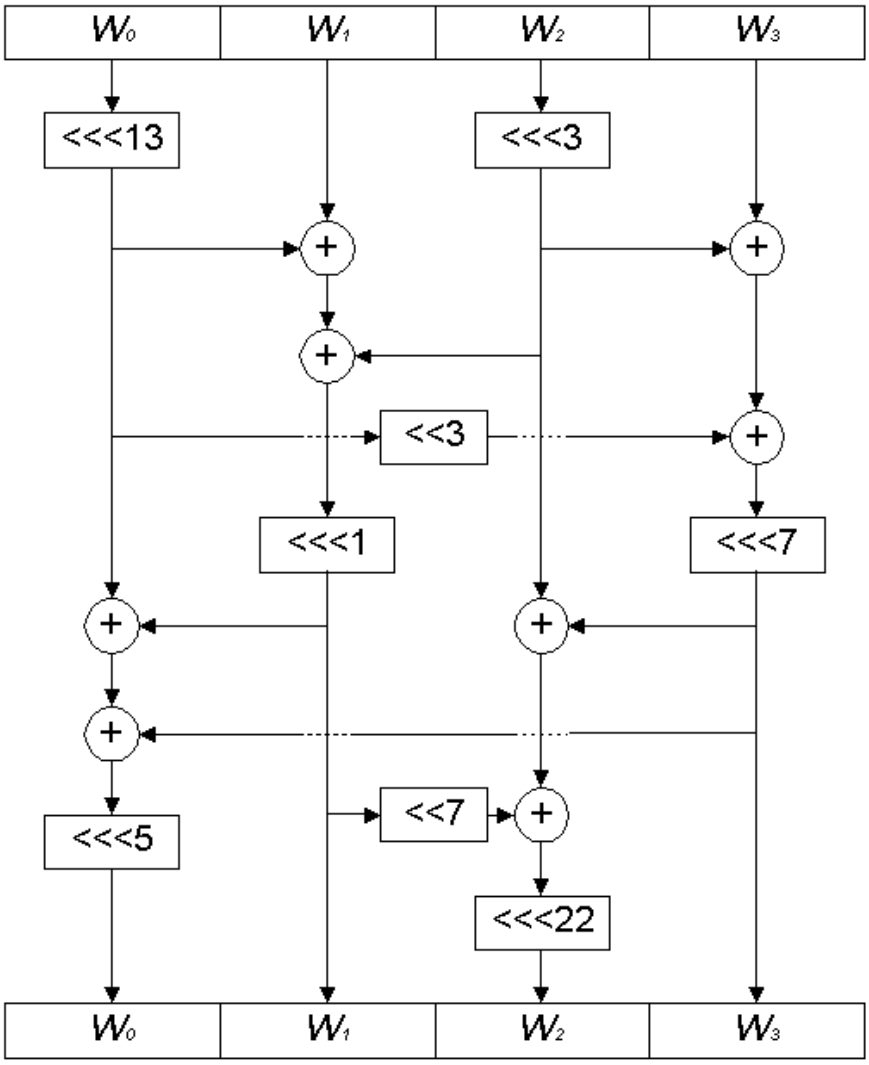


Рисунок 1.10 - Линейное преобразование в алгоритме Serpent.

Линейное преобразование может быть реализовано как в виде приведенной таблицы, так и с помощью следующих вычислений, на рисунке 1.10:

$$W_0 = W_0 <<< 13,$$

$$W_2 = W_2 <<< 3,$$

$$W_1 = W_1 \oplus W_0 \oplus W_2,$$
$$W_3 = W_3 \oplus W_2 \oplus (W_0 << 3),$$
$$W_1 = W_1 <<< 1,$$
$$W_3 = W_3 <<< 7,$$
$$W_0 = W_0 \oplus W_1 \oplus W_3,$$
$$W_2 = W_2 \oplus W_3 \oplus (W_1 << 7),$$
$$W_0 = W_0 <<< 5,$$
$$W_2 = W_2 <<< 22,$$

где W_n — **n**-е 32-битное слово обрабатываемого блока,

<< и <<< — соответственно, операции сдвига и циклического сдвига влево на указанное число бит. Основным критерием при разработке линейного преобразования было максимальное ускорение влияния каждого бита входного значения и ключа шифрования на каждый бит шифртекста. Как пишут авторы алгоритма , такое влияние достигается уже после трех раундов алгоритма Serpent.

Таким образом, преобразования, выполняемые в каждом раунде алгоритма, можно представить следующей формулой:

$$Y = L(S'_{(i \bmod 8)}(X \oplus K_i)),$$

где **X** и **Y** — соответственно, входное и выходное значения обрабатываемого блока,

i — номер раунда (от 0 до 31),

K_i — ключ i-го раунда,

L — линейное преобразование,

а символом **S'** обозначено параллельное использование 32 соответствующих таблиц замен.

Последний раунд отличается от предыдущих тем, что вместо линейного преобразования в нем выполняется дополнительное наложение ключа; здесь используется 33-й фрагмент расширенного ключа K_{32}:

$$Y = S'_7(X \oplus K_{31}) \oplus K_{32}$$

1.2.2 Расшифрование

Расшифрование производится выполнением обратных операций в обратной последовательности. В частности, вместо таблиц замен $S_0...S_7$ применяются в обратной последовательности инверсные таблицы замен $InvS_0...InvS_7$:

$InvS_0$	13	3	11	0	10	6	5	12	1	14	4	7	15	9	8	2
$InvS_1$	5	8	2	14	15	6	12	3	11	4	7	9	1	13	10	0
$InvS_2$	12	9	15	4	11	14	1	2	0	3	6	13	5	8	10	7

$InvS_3$	0	9	10	7	11	14	6	13	3	5	12	2	4	8	15	1
$InvS_4$	5	0	8	3	10	9	7	14	2	12	11	6	4	15	13	1
$InvS_5$	8	15	2	9	4	1	13	14	11	6	5	3	7	12	10	0
$InvS_6$	15	10	1	13	5	3	6	0	4	9	14	7	2	12	8	11
$InvS_7$	3	0	6	13	9	14	15	8	5	12	11	7	10	1	4	2

Аналогичным образом вместо «прямого» линейного преобразования используется обратное, которое определяется следующей таблицей (принцип действия обратного преобразования аналогичен прямому и описан выше):

53, 55, 72	1, 5, 20, 90	15, 102	3, 31, 90
57, 59, 76	5, 9, 24, 94	19, 106	7, 35, 94
61, 63, 80	9, 13, 28, 98	23, 110	11, 39, 98
65, 67, 84	13, 17, 32, 102	27, 114	1, 3, 15, 20, 43, 102
69, 71, 88	17, 21, 36, 106	1, 31, 118	5, 7, 19, 24, 47, 106
73, 75, 92	21, 25, 40, 110	5, 35, 122	9, 11, 23, 28, 51, 110
77, 79, 96	25, 29, 44, 114	9, 39, 126	13, 15, 27, 32, 55, 114
81, 83, 100	1, 29, 33, 48, 118	2, 13, 43	1, 17, 19, 31, 36, 59, 118
85, 87, 104	5, 33, 37, 52, 122	6, 17, 47	5, 21, 23, 35, 40, 63, 122
89, 91, 108	9, 37, 41, 56, 126	10, 21, 51	9, 25, 27, 39, 44, 67, 126
93, 95, 112	2, 13, 41, 45, 60	14, 25, 55	2, 13, 29, 31, 43, 48, 71
97, 99, 116	6, 17, 45, 49, 64	18, 29, 59	6, 17, 33, 35, 47, 52, 75
101, 103, 120	10, 21, 49, 53, 68	22, 33, 63	10, 21, 37, 39, 51, 56, 79
105, 107, 124	14, 25, 53, 57, 72	26, 37, 67	14, 25, 41, 43, 55, 60, 83
0, 109, 111	18, 29, 57, 61, 76	30, 41, 71	18, 29, 45, 47, 59, 64, 87
4, 113, 115	22, 33, 61, 65, 80	34, 45, 75	22, 33, 49, 51, 63, 68, 91
8, 117, 119	26, 37, 65, 69, 84	38, 49, 79	26, 37, 53, 55, 67, 72, 95

12, 121, 123	30, 41, 69, 73, 88	42, 53, 83	30, 41, 57, 59, 71, 76, 99
16, 125, 127	34, 45, 73, 77, 92	46, 57, 87	34, 45, 61, 63, 75, 80, 103
1, 3, 20	38, 49, 77, 81, 96	50, 61, 91	38, 49, 65, 67, 79, 84, 107
5, 7, 24	42, 53, 81, 85, 100	54, 65, 95	42, 53, 69, 71, 83, 88, 111
9, 11, 28	46, 57, 85, 89, 104	58, 69, 99	46, 57, 73, 75, 87, 92, 115
13, 15, 32	50, 61, 89, 93, 108	62, 73, 103	50, 61, 77, 79, 91, 96, 119
17, 19, 36	54, 65, 93, 97, 112	66, 77, 107	54, 65, 81, 83, 95, 100, 123
21, 23, 40	58, 69, 97, 101, 116	70, 81, 111	58, 69, 85, 87, 99, 104, 127
25, 27, 44	62, 73, 101, 105, 120	74, 85, 115	3, 62, 73, 89, 91, 103, 108
29, 31, 48	66, 77, 105, 109, 124	78, 89, 119	7, 66, 77, 93, 95, 107, 112
33, 35, 52	0, 70, 81, 109, 113	82, 93, 123	11, 70, 81, 97, 99, 111, 116
37, 39, 56	4, 74, 85, 113, 117	86, 97, 127	15, 74, 85, 101, 103, 115, 120
41, 43, 60	8, 78, 89, 117, 121	3, 90	19, 78, 89, 105, 107, 119, 124
45, 47, 64	12, 82, 93, 121, 125	7, 94	0, 23, 82, 93, 109, 111, 123
49, 51, 68	1, 16, 86, 97, 125	11, 98	4, 27, 86, 97, 113, 115, 127

1.2.3 Процедура расширения ключа

Согласно требованиям конкурса AES, Serpent поддерживает три размера ключа шифрования: 128, 192 и 256 бит. Однако, фактически, ключ алгоритма Serpent может иметь произвольный размер, меньший 256. Такой «неполный» ключ перед выполнением его расширения дополняется по следующему правилу:

- Ключ дополняется одним единичным битом справа.
- Затем ключ дополняется справа нулевыми битами до достижения 256-битного размера.

Только после этого производится процедура расширения ключа, состоящая из следующих шагов:

- 256-битный дополненный (при необходимости) ключ шифрования представляется в виде 8 32-битных слов, обозначаемых $\mathbf{w_{-8}...w_{-1}}$.

- Данные слова используются в качестве исходных значений для вычисления промежуточного ключа — последовательности $w_0 ... w_{131}$:

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) <<< 11,$$

где ϕ — округленная дробная часть золотого сечения $(\sqrt{5} + 1) / 2$, т.е. 9E3779B9 в шестнадцатеричной записи.

- Вычисляются ключи раунда с использованием таблиц замен $S_0 ... S_7$ (аналогично описанному выше их использованию в алгоритме Serpent):

$$K_0 = S_3(\{w_0, w_1, w_2, w_3\}),$$
$$K_1 = S_2(\{w_4, w_5, w_6, w_7\}),$$
$$K_2 = S_1(\{w_8, w_9, w_{10}, w_{11}\}),$$
$$K_3 = S_0(\{w_{12}, w_{13}, w_{14}, w_{15}\}),$$
$$K_4 = S_7(\{w_{16}, w_{17}, w_{18}, w_{19}\}),$$
$$K_5 = S_6(\{w_{20}, w_{21}, w_{22}, w_{23}\}),$$
$$K_6 = S_5(\{w_{24}, w_{25}, w_{26}, w_{27}\}),$$
$$K_7 = S_4(\{w_{28}, w_{29}, w_{30}, w_{31}\}),$$
$$...$$
$$K_{31} = S_4(\{w_{124}, w_{125}, w_{126}, w_{127}\}),$$
$$K_{32} = S_3(\{w_{128}, w_{129}, w_{130}, w_{131}\}).$$

1.3 Алгоритм шифрования Twofish

Аналогично алгоритму Serpent, Twofish разработан коллективом известных криптологов под руководством Брюса Шнайера (Bruce Schneier) — автора множества работ в области криптологии (в т. ч. известнейшей книги), разработчика известного алгоритма шифрования Blowfish и основателя американской компании Counterpane Systems, являющейся одним из мировых лидеров в области разработки средств криптографической защиты информации. Кроме него, в разработке алгоритма принимали участие Джон Келси (John Kelsey), Крис Холл (Chris Hall) и Нильс Фергюсон (Niels Ferguson) из той же компании Counterpane Systems, а также Дуг Уайтинг (Doug Whiting) из Hifn Incorporated (разработка аппаратных средств защиты Internet-коммуникаций) и Дэвид Вагнер (David Wagner) изУниверситета штата Калифорния.

1.3.1 Структура алгоритма

Twofish разбивает шифруемые данные на четыре 32-битных субблока (обозначим их **A**, **B**, **C**,**D**), над которыми производится 16 раундов

преобразований, в каждом из которых выполняются следующие операции, показано на рисунке 1.11 :

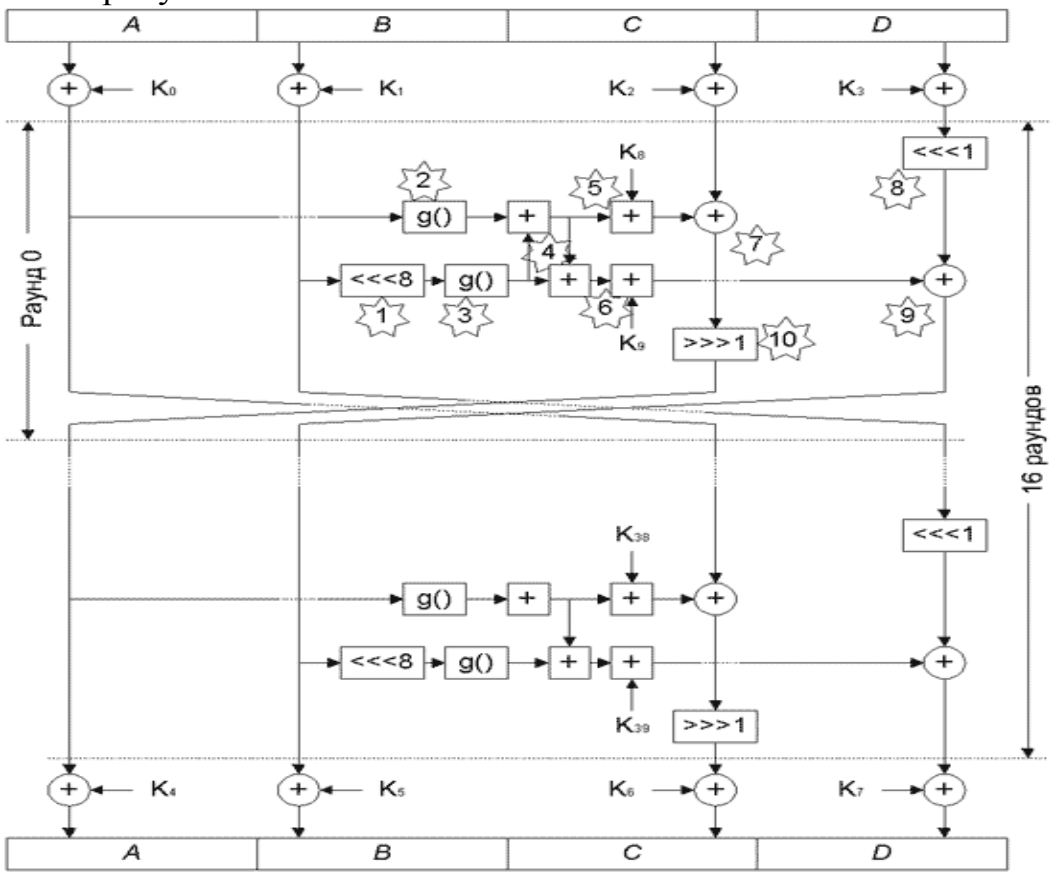


Рисунок 1.11 - Структура алгоритма Twofish.

1. Содержимое субблока **B** циклически сдвигается влево на 8 бит.
2. Субблок **A** обрабатывается операцией **g()**, которая будет подробно описана ниже.
3. Субблок **B** также обрабатывается операцией **g()**.
4. Субблок **B** накладывается на **A** с помощью сложения по модулю 2^{32}, после чего аналогичным образом выполняется наложение субблока **A** на субблок **B**.
5. Фрагмент расширенного ключа $\mathbf{K_{2r+8}}$ (где **r** — номер текущего раунда, начиная с 0) складывается с субблоком A по модулю 2^{32}.
6. Аналогично предыдущему шагу, $\mathbf{K_{2r+9}}$ накладывается на субблок **B**.
7. Субблок **A** накладывается на **C** операцией XOR.
8. Содержимое субблока **D** циклически сдвигается влево на 1 бит.
9. Субблок **B** накладывается на **D** операцией XOR.
10. Содержимое субблока **C** циклически сдвигается вправо на 1 бит.

$$\mathbf{B = B <<< 8,}$$
$$\mathbf{A = g(A),}$$
$$\mathbf{B = g(B),}$$
$$\mathbf{A = A + B \bmod 2^{32},}$$

$$B = A + B \bmod 2^{32},$$
$$A = A + K_{2r+8} \bmod 2^{32},$$
$$B = B + K_{2r+9} \bmod 2^{32},$$
$$C = C \text{ A } A,$$
$$D = D <<< 1,$$
$$D = D \text{ A } B,$$
$$C = C >>> 1.$$

Операция **g()** обрабатывает 32-битных входной субблок выполнением перечисленных ниже шагов, показано на рисунке 1.12 :

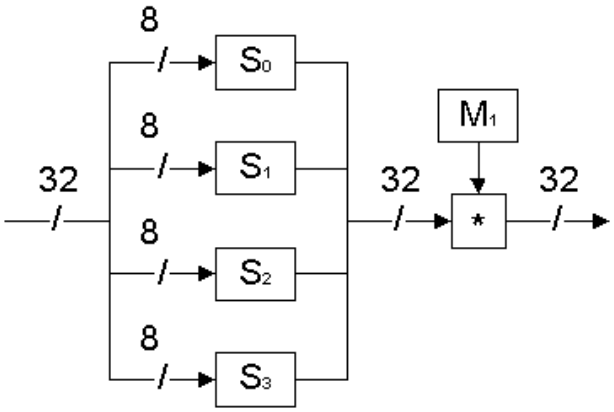


Рисунок 1.12 - Функция **g()** алгоритма Twofish.

1. Субблок делится на 4 фрагмента по 8 бит каждый.
2. Фрагменты прогоняются через таблицы замен 8 * 8 бит $\mathbf{S_0...S_3}$. Таблицы замен вычисляются динамически и зависят от ключа шифрования; алгоритм построения таблиц замен будет подробно описан ниже.
3. Результат замен (обозначим его $\mathbf{s_0...s_3}$) умножается на фиксированную матрицу $\mathbf{M_1}$:

$$\begin{pmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{pmatrix} = M_1 * \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix},$$

где $\mathbf{g_0...g_3}$ — байты выходного значения функции **g()**.
Умножение выполняется в конечном поле GF(2^8) с образующим полиномом $\mathbf{x^8 + x^6 + x^5 + x^3 + 1}$.
Матрица M_1 определена следующим образом (здесь и далее в таблицах указаны шестнадцатеричные значения):

01	EF	5B	5B

5B	EF	EF	01
EF	5B	01	EF
EF	01	EF	5B

В конце каждого раунда, за исключением последнего, субблоки **A** (значение до описанной выше обработки) и **C** меняются местами; субблоки **B** (значение до обработки) и **D** также меняются местами.
Перед первым раундом выполняется *входное отбеливание* — наложение операцией XOR на обрабатываемые субблоки четырех фрагментов расширенного ключа $\mathbf{K_0...K_3}$. Аналогично выполняется выходное отбеливание — после последнего раунда с использованием подключей$\mathbf{K_4...K_7}$.
Как видно, алгоритм Twofish имеет существенно более сложную структуру, чем Serpent, что компенсируется вдвое меньшим количеством раундов.

1.3.2 Процедура расширения ключа

Процедура расширения ключа формирует 40 32-битных подключей для использования в 16 раундах алгоритма и для выполнения операций отбеливания. Кроме того, на основе ключа шифрования вычисляются таблицы замен $\mathbf{S_0...S_3}$.
Аналогично алгоритму Serpent, Twofish использует ключи шифрования любого размера до 256 бит включительно. Однако, дополнение ключа выполняется несколько иначе: исходный ключ, при необходимости, дополняется нулевыми битами до ближайшего стандартного размера, т.е. до 128, 192 или 256 бит. Процедура расширения ключа обрабатывает дополненный таким образом ключ.
Сначала выполняется предварительная обработка ключа, включающая в себя следующие шаги:

1. Выполняется инициализация переменных, участвующих в дальнейших расчетах:

$$\mathbf{k = N / 64,}$$

где **N** — размер дополненного ключа шифрования в битах, т.е. **k** принимает значение 2, 3 или 4. Ключ шифрования представляется в виде **8k** байт $\mathbf{m_0...m_{8k-1}}$ или в виде **2k** 32-битных слов, обозначаемых как $\mathbf{M_0...M_{2k-1}}$.

2. Формируются 3 массива, каждый из которых состоит из **k** 32-битных слов:

$$\mathbf{M_e = (M_0, M_2, ..., M_{2k-2}),}$$
$$\mathbf{M_o = (M_1, M_3, ..., M_{2k-1}),}$$
$$\mathbf{V = (V_{k-1}, V_{k-2}, ..., V_0),}$$

где:

$$\mathbf{v_i} = \sum_{j=0}^{3} v_{i,j} * 2^{8j},$$

$$\begin{pmatrix} v_{i,0} \\ v_{i,1} \\ v_{i,2} \\ v_{i,3} \end{pmatrix} = M_2 * \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+} \\ m_{8i+7} \end{pmatrix}.$$

Матрица $\mathbf{M_2}$ определена следующим образом:

01	A4	55	87	5A	58	DB	9E
A4	56	82	F3	1E	C6	68	E5
02	A1	FC	C1	47	AE	3D	19
A4	55	87	5A	58	DB	9E	03

Генерация подключей $\mathbf{k_0...k_{39}}$ производится на основе вычисленных на предварительном этапе массивов $\mathbf{M_e}$ и $\mathbf{M_o}$ следующим образом:

$$\mathbf{k_{2i} = A_i + B_i \bmod 2^{32},}$$
$$\mathbf{k_{2i+1} = (A_i + 2B_i \bmod 2^{32}) <<< 9,}$$

где $\mathbf{i} = 0...19$, а $\mathbf{A_i}$ и $\mathbf{B_i}$ — промежуточные величины, вычисляемые так:

$$\mathbf{A_i = h(2i\rho, M_e),}$$
$$\mathbf{B_i = h((2i + 1)\rho, M_o) <<< 8.}$$

Константа $\boldsymbol{\rho}$ определена следующим образом:

$$\boldsymbol{\rho = 2^{24} + 2^{16} + 2^8 + 1,}$$

а функция **h()** заслуживает подробного описания.

Данная функция схематично представлена на рисунке 1.13.

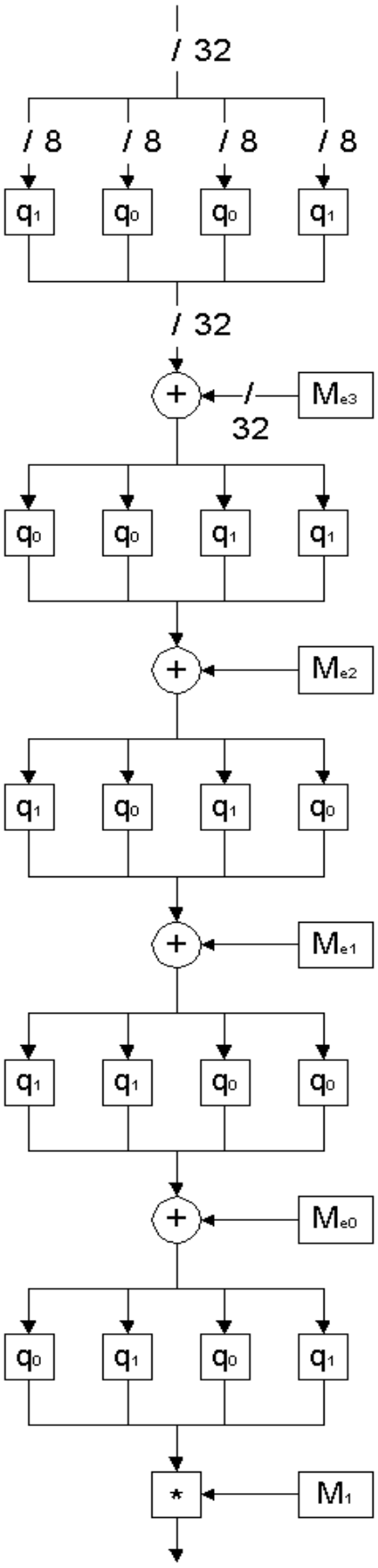


Рисунок 1.13 - Функция **h()** алгоритма Twofish.

Она выполняется в несколько шагов, состав которых зависит от размера дополненного ключа в 64-битных фрагментах, т.е. от описанного выше значения **k**. В качестве параметров функция**h()** принимает 32-битное слово и массив 32-битных слов размерностью **k**. Последовательность выполнения такова:

1. Входное слово делится на 4 8-битных фрагмента, которые прогоняются через специфические операции замены $\mathbf{q_0}$ и $\mathbf{q_1}$ (как показано на рис. 5). Результат замены объединяется в 32-битное слово, которое складывается с третьим словом входного массива (на рис. 5 в качестве входного массива показан массив $\mathbf{M_e}$). Данный шаг не выполняется, если $\mathbf{k} < 4$ (т.е. $\mathbf{k} = 2$ или $\mathbf{k} = 3$). На рис. 5 показаны все возможные шаги функции **h()**.
2. Результат предыдущего шага или входное слово обрабатывается аналогичным образом (с другой последовательностью применения $\mathbf{q_0}$ и $\mathbf{q_1}$, которая является различной для каждого шага — см. рис. 5), но складывается со вторым словом входного массива. Шаг 2 не выполняется, если $\mathbf{k} = 2$.
3. Выполняются всегда и подразумевают обработку результата предыдущего шага (или входного слова — для шага 3 и $\mathbf{k} = 2$), аналогичную предыдущим шагам с использованием, соответственно, первого и нулевого слов входного массива.
4. Как и на предыдущих шагах, применяются замены $\mathbf{q_0}$ и $\mathbf{q_1}$, после чего выполняется следующее преобразование:

$$\begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{pmatrix} = M_1 * \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix},$$

$$\mathbf{H} = \sum_{i=0}^{3} h_i * 2^{8i},$$

где $\mathbf{y_0 ... y_3}$ — байты результата выполнения замен на шаге 5, а матрица $\mathbf{M_1}$ была описана выше; **H** — выходное значение функции **h()**. Операции $\mathbf{q_0}$ и $\mathbf{q_1}$ представляют собой не табличные замены 8 * 8 бит, а вычисляют выходные значения с использованием нескольких таблиц замен 4 * 4 следующим образом:

$$\mathbf{a_0 = \lfloor x / 16 \rfloor,}$$
$$\mathbf{b_0 = x \bmod 16,}$$
$$\mathbf{a_1 = a_0 \ A \ b_0,}$$

$$b_1 = a_0 \text{ A } (b_0 >>>_4 1) \text{ A } 8a_0 \bmod 16,$$
$$a_2 = t_0(a_1),$$
$$b_2 = t_1(b_1),$$
$$a_3 = a_2 \text{ A } b_2,$$
$$b_3 = a_2 \text{ A } (b_2 >>>_4 1) \text{ A } 8a_2 \bmod 16,$$
$$a_4 = t_2(a_3),$$
$$b_4 = t_3(b_3),$$
$$y = 16b_4 + a_4,$$

где **x** — входное значение,
y — выходное значение,
$>>>_4$ — операция циклического вращения вправо 4-битных величин, т.е. раздельное вращение каждого нибла обрабатываемого байта;
$\mathbf{t_i}$ — табличные замены, различные для $\mathbf{q_0}$ и $\mathbf{q_1}$; для $\mathbf{q_0}$ таблицы выглядят так:

$\mathbf{t_0}$	8	1	7	D	6	F	3	2	0	B	5	9	E	C	A	4
$\mathbf{t_1}$	E	C	B	8	1	2	3	5	F	4	A	6	7	0	9	D
$\mathbf{t_2}$	B	A	5	E	6	D	9	0	C	8	F	3	2	4	7	1
$\mathbf{t_3}$	D	7	F	4	1	2	6	E	9	B	3	0	8	5	C	A

Выходное значение таблицы берется из позиции, соответствующей входному значению; например, $\mathbf{t_1}$ заменяет 0 на E, 1 на C и т.д.
Таблицы для $\mathbf{q_1}$ определены следующим образом:

$\mathbf{t_0}$	2	8	B	D	F	7	6	E	3	1	9	4	0	A	C	5
$\mathbf{t_1}$	1	E	2	B	4	C	3	7	6	D	A	5	F	9	0	8
$\mathbf{t_2}$	4	C	7	5	1	6	9	A	0	E	D	8	2	B	3	F
$\mathbf{t_3}$	B	9	5	1	C	3	D	E	6	4	7	F	2	0	8	A

Осталось описать зависимость от ключа таблиц замен $\mathbf{S_0 ... S_3}$. Фактически, в алгоритме шифрования вместо описанной выше функции **g()** используется функция **h()**, использующая в качестве входного значения 32-битный субблок **A** или **B**, а в качестве входного массива — описанный ранее массив **V**. Таким образом, замена выполняется на основе тех же таблиц$\mathbf{t_0 ... t_3}$, которые используются в процедуре расширения ключа.
Видно, что процедура расширения ключа является сложной и занимает несравнимо больше времени, чем шифрование одного блока данных. Авторы алгоритма в его спецификации описали несколько возможных вариантов

оптимизации процедуры расширения ключа, один из которых можно выбрать в конкретной реализации алгоритма, исходя из следующих параметров:

- доступных объемов энергонезависимой и оперативной памяти,
- предполагаемой частоты смены ключа при шифровании данных,
- требований к скорости шифрования блоков данных.

Общий принцип оптимизации состоит в поисках требуемого компромисса между скоростью шифрования и временем выполнения процедуры расширения ключа: чем быстрее нужно шифровать, тем дольше будет выполняться расширение ключа, и наоборот.

1.3.3 Twofish-FK

Помимо описанного выше «стандартного» варианта алгоритма Twofish, авторы предложили еще и Twofish Family Key (или Twofish-FK) — шаблон для формирования на основе Twofish вариантов алгоритма, несовместимых между собой и предназначенных, таким образом, для ограниченных применений, например, в рамках конкретных организаций для защиты внутреннего документооборота.
Несовместимость вариантов достигается путем применения дополнительного ключа (**FK**), константного для всех субъектов, применяющих конкретный из вариантов алгоритма Twofish-FK. Т.е. конкретное значение используемого **FK** формирует некий «контур совместимости» криптосредств, реализующих Twofish-FK.
Дополнительный ключ **FK** «подмешивается» на этапе расширения ключа следующим образом:

1. Инициализируются переменные, используемые в дальнейших операциях (данный шаг выполняется однократно для конкретного **FK**):

$$T_0 = FK,$$
$$T_1 = (E_{FK}(0) \mid E_{FK}(1)) \text{ A } FK,$$
$$T_2 = E_{FK}(2),$$
$$T_3 = E_{FK}(3),$$
$$T_4 = E_{FK}(4) \bmod 2^{64},$$

где $E_{FK}(N)$ — результат зашифрования алгоритмом Twofish на ключе **FK** 128-битного блока, первый байт которого содержит значение **N**, а остальные байты обнулены,
| — операция конкатенации.
Стоит отметить, что переменные T_0 и T_1 имеют размерность 256 бит, T_2 и T_3 — по 128 бит, а T_4 — 64 бита.

2. Наложение ключа **FK**, выполняется однократно для каждого ключа шифрования, используемого совместно с данным **FK**:

- Перед выполнением процедуры расширения ключа шифрования выполняется наложение $\mathbf{T_0}$ на ключ шифрования операцией XOR. Если ключ шифрования имеет размер, меньший 256 бит, то используется необходимое количество бит $\mathbf{T_0}$.
- После выполнения процедуры расширения ключа операциями XOR $\mathbf{T_2}$ накладывается на подключи $\mathbf{K_0...K_3}$, $\mathbf{T_3}$ — на подключи $\mathbf{K_4...K_7}$, а $\mathbf{T_4}$ — на каждую пару подключей, используемых в раундах шифрования.
- Перед использованием табличных замен выполняется наложение операцией XOR требуемого количества бит $\mathbf{T_1}$ на ключ шифрования.

Криптостойкость алгоритмов, построенных на основе Twofish-FK, по словам авторов алгоритма, полностью эквивалентна криптостойкости стандартного Twofish, поскольку подобное наложение дополнительного ключа никак не влияет на основные параметры алгоритма.

1.4 Алгоритм шифрования MARS

Алгоритм MARS был разработан коллективом криптологов из корпорации IBM. Именно IBM в свое время разработала семейство алгоритмов Lucifer, которое легло в основу прошлого стандарта шифрования США — DES. Из авторов Lucifer в разработке алгоритма MARS принял участие Дон Копперсмит (Don Coppersmith), известный также и другими работами в области криптологии.

По правилам конкурса AES разрешалось вносить незначительные изменения в участвовавшие в конкурсе алгоритмы в течение первого раунда конкурса. Пользуясь этим правилом, авторы алгоритма MARS изменили процедуру расширения ключа, в результате чего существенно снизились требования, предъявляемые алгоритмом к энергонезависимой и оперативной памяти. Рассмотрим здесь именно модифицированную версию алгоритма.

1.4.1 Структура алгоритма

Исходя из двух следующих предположений:

1. Многие известные криптоаналитические методы отличают первый и последний раунды алгоритма (или несколько первых и/или несколько последних раундов) от остальных и применяют к ним другие приемы, нежели к «центральным» раундам алгоритма. Таким образом, различные раунды алгоритма шифрования играют различное значение в обеспечиваемой алгоритмом криптостойкости.

2. Скорее всего, алгоритм с гетерогенной структурой будет лучше противостоять криптоаналитическим методам будущего, чем алгоритм, все раунды которого идентичны.

- разработчики алгоритма MARS придали ему сильно гетерогенную структуру — раунды алгоритма весьма различаются между собой. Алгоритм MARS можно описать следующим образом на рисунке 1.14:

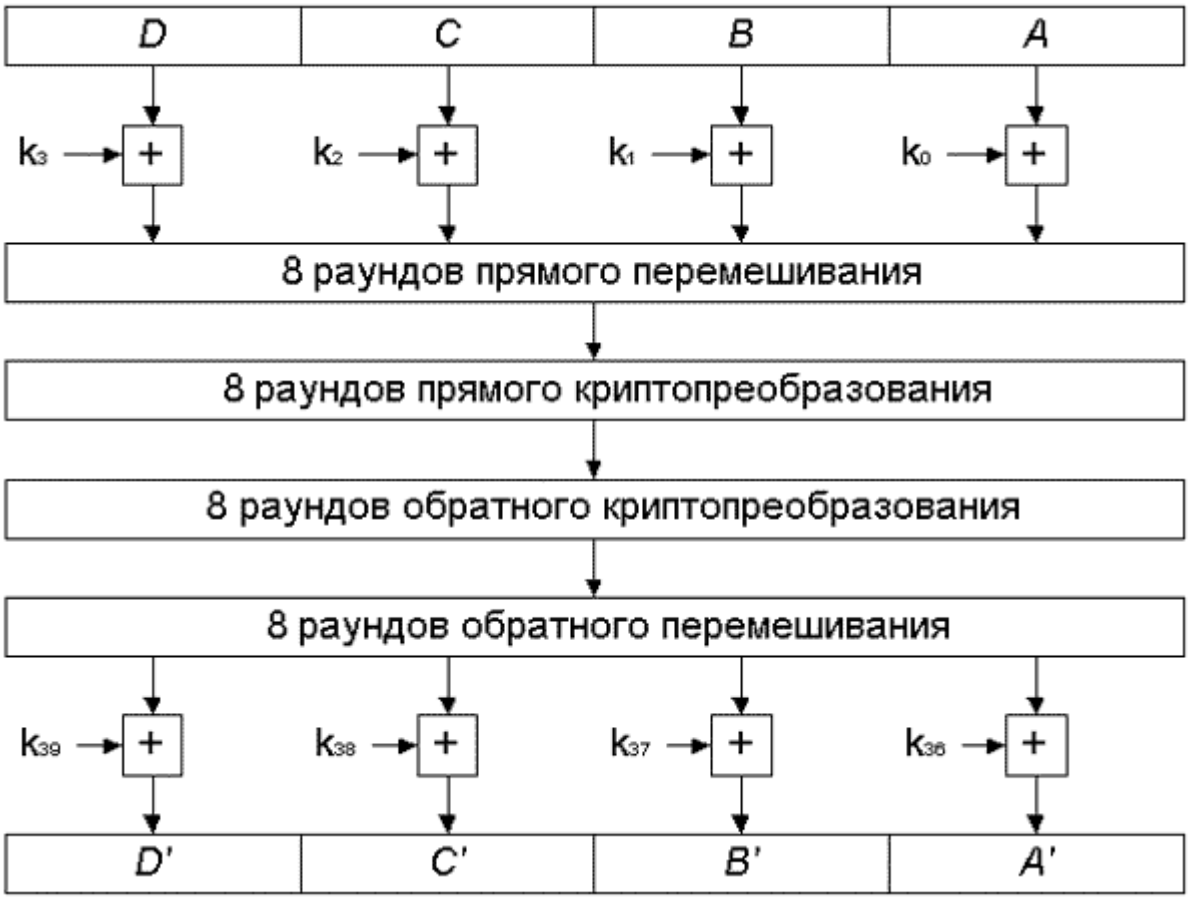


Рисунок 1.14 - Структура алгоритма MARS.

1. Предварительное наложение ключа: на 32-битные субблоки **A**, **B**, **C**, **D** накладываются 4 фрагмента расширенного ключа $\mathbf{k_0...k_3}$ операцией сложения по модулю 2^{32}.
2. Выполняются 8 раундов прямого перемешивания (без участия ключа шифрования).
3. Выполняются 8 раундов прямого криптопреобразования.
4. Выполняются 8 раундов обратного криптопреобразования. Этапы 3 и 4 называются «криптографическим ядром» алгоритма MARS.
5. Выполняются 8 раундов обратного перемешивания, также без участия ключа шифрования.
6. Финальное наложение фрагментов расширенного ключа $\mathbf{k_{36}...k_{39}}$ операцией вычитания по модулю 2^{32}.

Алгоритм представляет собой расширенную сеть Фейстеля. В каждом раунде выполняется обработка одного из субблоков и наложение результатов обработки на остальные субблоки, после чего субблоки меняются местами. Конкретные преобразования зависят от типа раунда и будут рассмотрены ниже. Кроме того, между раундами могут выполняться различные дополнительные действия, которые также будут описаны далее.

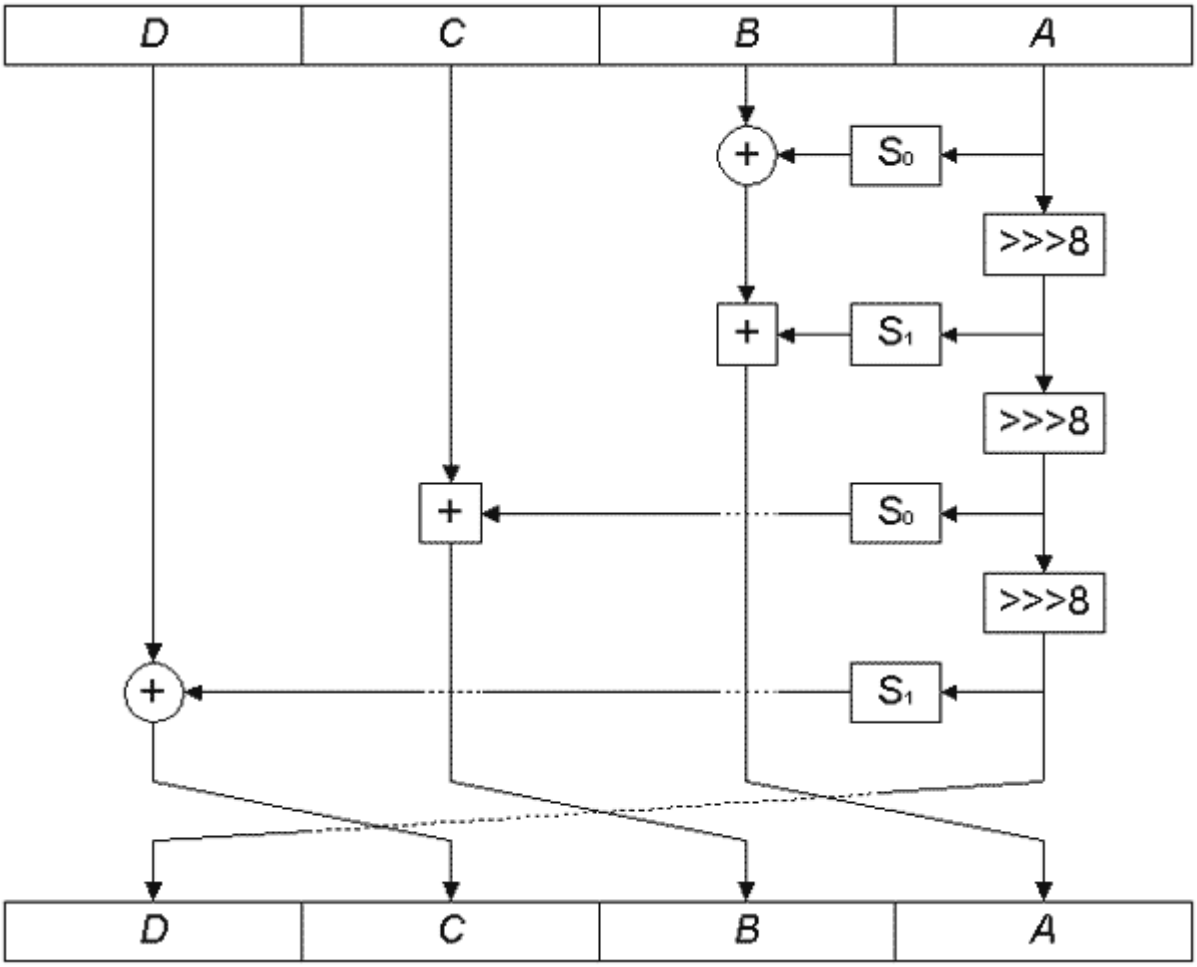


Рисунок 1.15 - Раунд прямого перемешивания алгоритма MARS.

Раунд прямого перемешивания показан на рисунке 1.15. Как видно из рисунка, в раунде выполняются следующие действия:

1. Значение субблока **A** прогоняется через таблицу замен $\mathbf{S_0}$ и накладывается на субблок **B**операцией XOR.
2. Исходное значение субблока **A** вращается на 8 бит вправо.
3. Результат предыдущего шага обрабатывается таблицей замен $\mathbf{S_1}$ и снова накладывается на субблок **B** операцией сложения по модулю 2^{32}.
4. Результат шага 2 вращается на 8 бит вправо.
5. Результат предыдущего шага обрабатывается таблицей замен $\mathbf{S_0}$ и накладывается на субблок **C** операцией сложения по модулю 2^{32}.
6. Результат шага 4 вращается на 8 бит вправо.

7. Результат предыдущего шага обрабатывается таблицей замен $\mathbf{S_1}$ и накладывается на субблок **D** операцией XOR.
8. Субблоки меняются местами, как показано на рисунке 1.15.

Кроме того, в некоторых раундах прямого перемешивания выполняются следующие дополнительные операции (не приведены на рис. 1.15):

1. В раундах 0 и 4 после шага 7 выполняется наложение значения субблока **D** на субблок **A**операцией сложения по модулю 2^{32}.
2. В раундах 1 и 5 субблок **B** аналогичным образом накладывается на субблок **A**.

По словам авторов алгоритма, эти операции существенно усиливают алгоритм MARS против дифференциального криптоанализа.

Таблицы замен $\mathbf{S_0}$ и $\mathbf{S_1}$ определены в спецификации алгоритма следующим образом:

S_0:

09D0C479	28C8FFE0	84AA6C39	9DAD7287	7DFF9BE3	D4268361
C96DA1D4	7974CC93	85D0582E	2A4B5705	1CA16A62	C3BD279D
0F1F25E5	5160372F	C695C1FB	4D7FF1E4	AE5F6BF4	0D72EE46
FF23DE8A	B1CF8E83	F14902E2	3E981E42	8BF53EB6	7F4BF8AC
83631F83	25970205	76AFE784	3A7931D4	4F846450	5C64C3F6
210A5F18	C6986A26	28F4E826	3A60A81C	D340A664	7EA820C4
526687C5	7EDDD12B	32A11D1D	9C9EF086	80F6E831	AB6F04AD
56FB9B53	8B2E095C	B68556AE	D2250B0D	294A7721	E21FB253
AE136749	E82AAE86	93365104	99404A66	78A784DC	B69BA84B
04046793	23DB5C1E	46CAE1D6	2FE28134	5A223942	1863CD5B
C190C6E3	07DFB846	6EB88816	2D0DCC4A	A4CCAE59	3798670D
CBFA9493	4F481D45	EAFC5CA5	DB1129D6	B0449E20	0F5407FB
6167D9A8	D1F45763	4DAA96C3	3BEC5958	ABABA014	B6CCD201
38D6279F	02682215	8F376CD5	092C237E	BFC56593	32889D2C
854B3E95	05BB5B43	7DCD5DCD	A02E926C	FAE527E5	36A1C330
3412E1AE	F257F462	3C4F1D71	30A2E809	68E5F551	9C61BA44
5DED0AB8	75CE09C8	9654F93E	698C0CCA	243CB3E4	2B062B97
0F3B8D9E	00E050DF	FC5D6166	E35F9288	C079550D	0591AEE8
8E531E74	75FE3578	2F6D829A	F60B21AE	95E8EB8D	6699486B
901D7D9B	FD6D6E31	1090ACEF	E0670DD8	DAB2E692	CD6D4365
E5393514	3AF345F0	6241FC4D	460DA3A3	7BCF3729	5BF1D1E0
14AAC070	1587ED55	3AFD7D3E	D2F29E01	29A9D1F6	EFB10C53
CF3B870F	B414935C	664465ED	024ACAC7	59A744C1	1D2936A7
DC580AA6	CF574CA8	040A7A10	6CD81807	8A98BE4C	ACCEA063
C33E92B5	D1E0E03D	B322517E	2092BD13	386B2C4A	52E8DD58
58656DFB	50820371	41811896	E337EF7E	D39FB119	C97F0DF6

68FEA01B	A150A6E5	55258962	EB6FF41B	D7C9CD7A	A619CD9E
BCF09576	2672C073	F003FB3C	4AB7A50B	1484126A	487BA9B1
A64FC9C6	F6957D49	38B06A75	DD805FCD	63D094CF	F51C999E
1AA4D343	B8495294	CE9F8E99	BFFCD770	C7C275CC	378453A7
7B21BE33	397F41BD	4E94D131	92CC1F98	5915EA51	99F861B7
C9980A88	1D74FD5F	B0A495F8	614DEED0	B5778EEA	5941792D
FA90C1F8	33F824B4	C4965372	3FF6D550	4CA5FEC0	8630E964
5B3FBBD6	7DA26A48	B203231A	04297514	2D639306	2EB13149
16A45272	532459A0	8E5F4872	F966C7D9	07128DC0	0D44DB62
AFC8D52D	06316131	D838E7CE	1BC41D00	3A2E8C0F	EA83837E
B984737D	13BA4891	C4F8B949	A6D6ACB3	A215CDCE	8359838B
6BD1AA31	F579DD52	21B93F93	F5176781	187DFDDE	E94AEB76
2B38FD54	431DE1DA	AB394825	9AD3048F	DFEA32AA	659473E3
623F7863	F3346C59	AB3AB685	3346A90B	6B56443E	C6DE01F8
8D421FC0	9B0ED10C	88F1A1E9	54C1F029	7DEAD57B	8D7BA426
4CF5178A	551A7CCA	1A9A5F08	FCD651B9	25605182	E11FC6C3
B6FD9676	337B3027	B7C8EB14	9E5FD030		

В качестве входного значения $\mathbf{S_0}$ (аналогично и $\mathbf{S_1}$) принимает 8 младших бит входного слова. Таблица меняет значение 0 на 09D0C479, значение 1 — на 28C8FFE0 и т. д.

S_1:

6B57E354	AD913CF7	7E16688D	58872A69	2C2FC7DF	E389CCC6
30738DF1	0824A734	E1797A8B	A4A8D57B	5B5D193B	C8A8309B
73F9A978	73398D32	0F59573E	E9DF2B03	E8A5B6C8	848D0704
98DF93C2	720A1DC3	684F259A	943BA848	A6370152	863B5EA3
D17B978B	6D9B58EF	0A700DD4	A73D36BF	8E6A0829	8695BC14
E35B3447	933AC568	8894B022	2F511C27	DDFBCC3C	006662B6
117C83FE	4E12B414	C2BCA766	3A2FEC10	F4562420	55792E2A
46F5D857	CEDA25CE	C3601D3B	6C00AB46	EFAC9C28	B3C35047
611DFEE3	257C3207	FDD58482	3B14D84F	23BECB64	A075F3A3
088F8EAD	07ADF158	7796943C	FACABF3D	C09730CD	F7679969
DA44E9ED	2C854C12	35935FA3	2F057D9F	690624F8	1CB0BAFD
7B0DBDC6	810F23BB	FA929A1A	6D969A17	6742979B	74AC7D05
010E65C4	86A3D963	F907B5A0	D0042BD3	158D7D03	287A8255
BBA8366F	096EDC33	21916A7B	77B56B86	951622F9	A6C5E650
8CEA17D1	CD8C62BC	A3D63433	358A68FD	0F9B9D3C	D6AA295B

FE33384A	C000738E	CD67EB2F	E2EB6DC2	97338B02	06C9F246
419CF1AD	2B83C045	3723F18A	CB5B3089	160BEAD7	5D494656
35F8A74B	1E4E6C9E	000399BD	67466880	B4174831	ACF423B2
CA815AB3	5A6395E7	302A67C5	8BDB446B	108F8FA4	10223EDA
92B8B48B	7F38D0EE	AB2701D4	0262D415	AF224A30	B3D88ABA
F8B2C3AF	DAF7EF70	CC97D3B7	E9614B6C	2BAEBFF4	70F687CF
386C9156	CE092EE5	01E87DA6	6CE91E6A	BB7BCC84	C7922C20
9D3B71FD	060E41C6	D7590F15	4E03BB47	183C198E	63EEB240
2DDBF49A	6D5CBA54	923750AF	F9E14236	7838162B	59726C72
81B66760	BB2926C1	48A0CE0D	A6C0496D	AD43507B	718D496A
9DF057AF	44B1BDE6	054356DC	DE7CED35	D51A138B	62088CC9
35830311	C96EFCA2	686F86EC	8E77CB68	63E1D6B8	C80F9778
79C491FD	1B4C67F2	72698D7D	5E368C31	F7D95E2E	A1D3493F
DCD9433E	896F1552	4BC4CA7A	A6D1BAF4	A5A96DCC	0BEF8B46
A169FDA7	74DF40B7	4E208804	9A756607	038E87C8	20211E44
8B7AD4BF	C6403F35	1848E36D	80BDB038	1E62891C	643D2107
BF04D6F8	21092C8C	F644F389	0778404E	7B78ADB8	A2C52D53
42157ABE	A2253E2E	7BF3F4AE	80F594F9	953194E7	77EB92ED
B3816930	DA8D9336	BF447469	F26D9483	EE6FAED5	71371235
DE425F73	B4E59F43	7DBE2D4E	2D37B185	49DC9A63	98C39D98
1301C9A2	389B1BBF	0C18588D	A421C1BA	7AA3865C	71E08558
3C5CFCAA	7D239CA4	0297D9DD	D7DC2830	4B37802B	7428AB54
AEEE0347	4B3FBB85	692F2F08	134E578E	36D9E0BF	AE8B5FCF
EDB93ECF	2B27248E	170EB1EF	7DC57FD6	1E760F16	B1136601
864E1B9B	D7EA7319	3AB871BD	CFA4D76F	E31BD782	0DBEB469
ABB96061	5370F85D	FFB07E37	DA50D0FB	EBC977B6	0B98B40F
3A4D0FE6	DF4FC26B	159CF22A	C298D6E2	2B78EF6A	61A94AC0
AB561187	14EEA0F0	DF0D4164	19AF70EE		

Структура раунда прямого криптопреобразования приведена на рисунке 1.16.

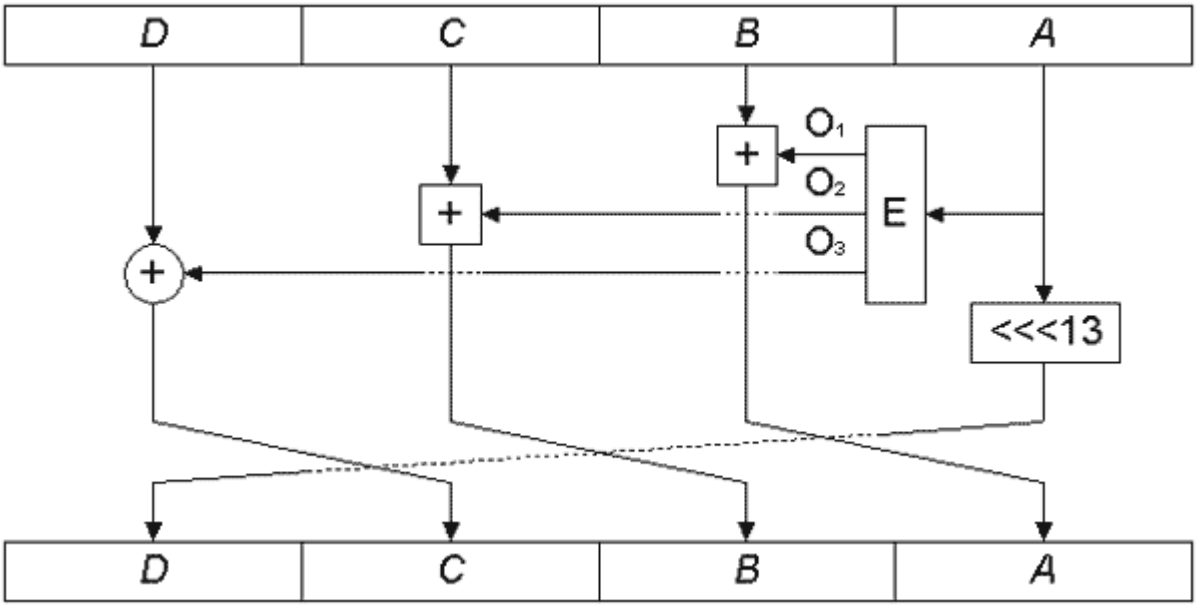


Рисунок 1.16 - Раунд прямого криптопреобразования алгоритма MARS.

Основой раунда является расширяющее криптопреобразование **E**, преобразующее 32-битное входное слово **A** в три выходных 32-битных значения, каждое из которых определенным образом накладывается на остальные субблоки , как на рисунке 1.16. После этого субблок **A** вращается влево на 13 бит, затем субблоки меняются местами аналогично раунду прямого перемешивания.
Преобразование **E** показано на рисунке 1.17.

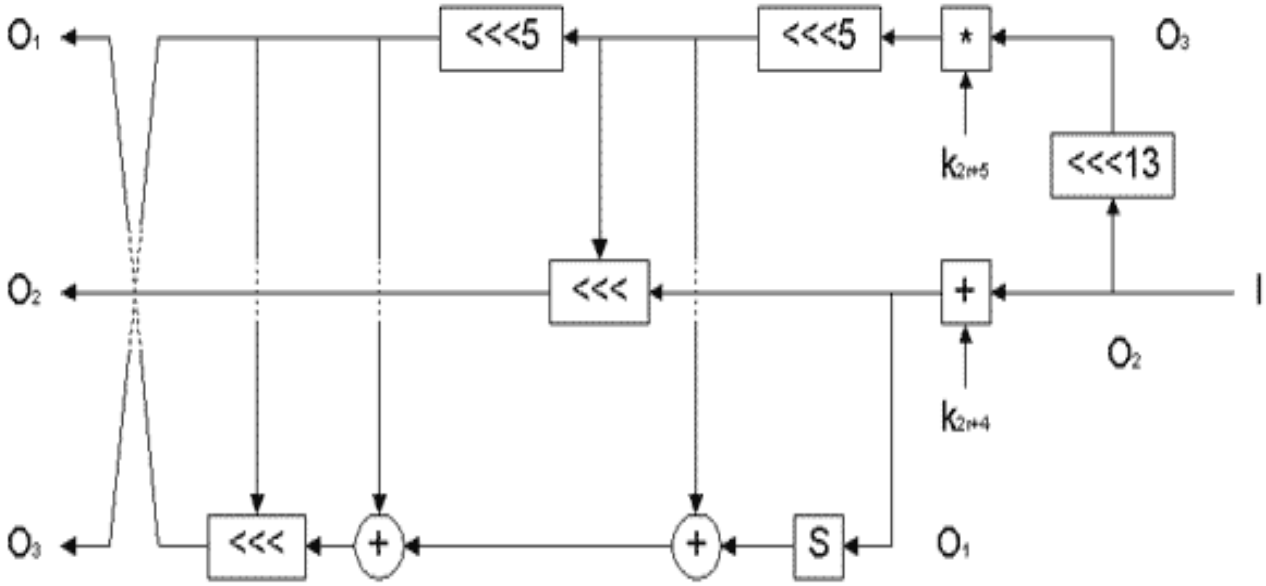


Рисунок 1.17 - Операция E алгоритма MARS.

Из входного значения формируются три потока $\mathbf{O_1...O_3}$, над которыми производятся следующие действия:

$$O_2 = I,$$
$$O_3 = O_2 <<< 13,$$
$$O_2 = O_2 + k_{2r+4} \bmod 2^{32},$$
$$O_3 = O_3 * k_{2r+5} \bmod 2^{32},$$
$$O_3 = O_3 <<< 5,$$
$$O_1 = S(O_2),$$
$$O_1 = O_1 \text{ A } O_3,$$
$$O_2 = O_2 <<< O_3',$$
$$O_3 = O_3 <<< 5,$$
$$O_1 = O_1 \text{ A } O_3,$$
$$O_1 = O_1 <<< O_3',$$

где **I** — входное значение,
r — номер текущего раунда, считая с 0-го (при нумерации раундов в данном случае учитываются только раунды криптоядра алгоритма),
S — табличная замена для операции **E**, представляет собой объединение описанных выше таблиц S_0 и S_1; объединенная таблица содержит 512 значений, выходное значение выбирается по значению 9 младших бит входного слова.
Стоит обратить внимание на то, что в преобразовании **E** используются операции вращения на переменное число бит; в этом случае запись O_3' обозначает, что число бит вращения определяется значением младших пяти бит текущего значения O_3.
Структура обратного криптораунда показана на рисунке 1.18.

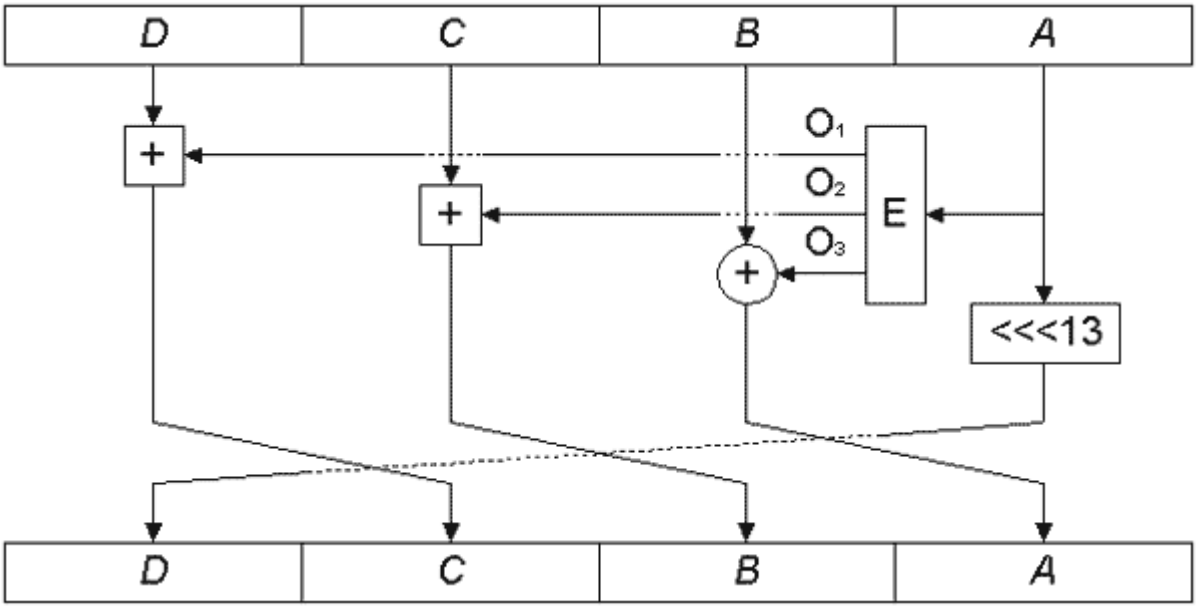


Рисунок 1.18 - Раунд обратного криптопреобразования алгоритма MARS.

От прямого криптораунда его отличает лишь измененный порядок наложения выходных значений преобразования **E** $O_1 \ldots O_3$ на слова **B**, **C** и **D**.

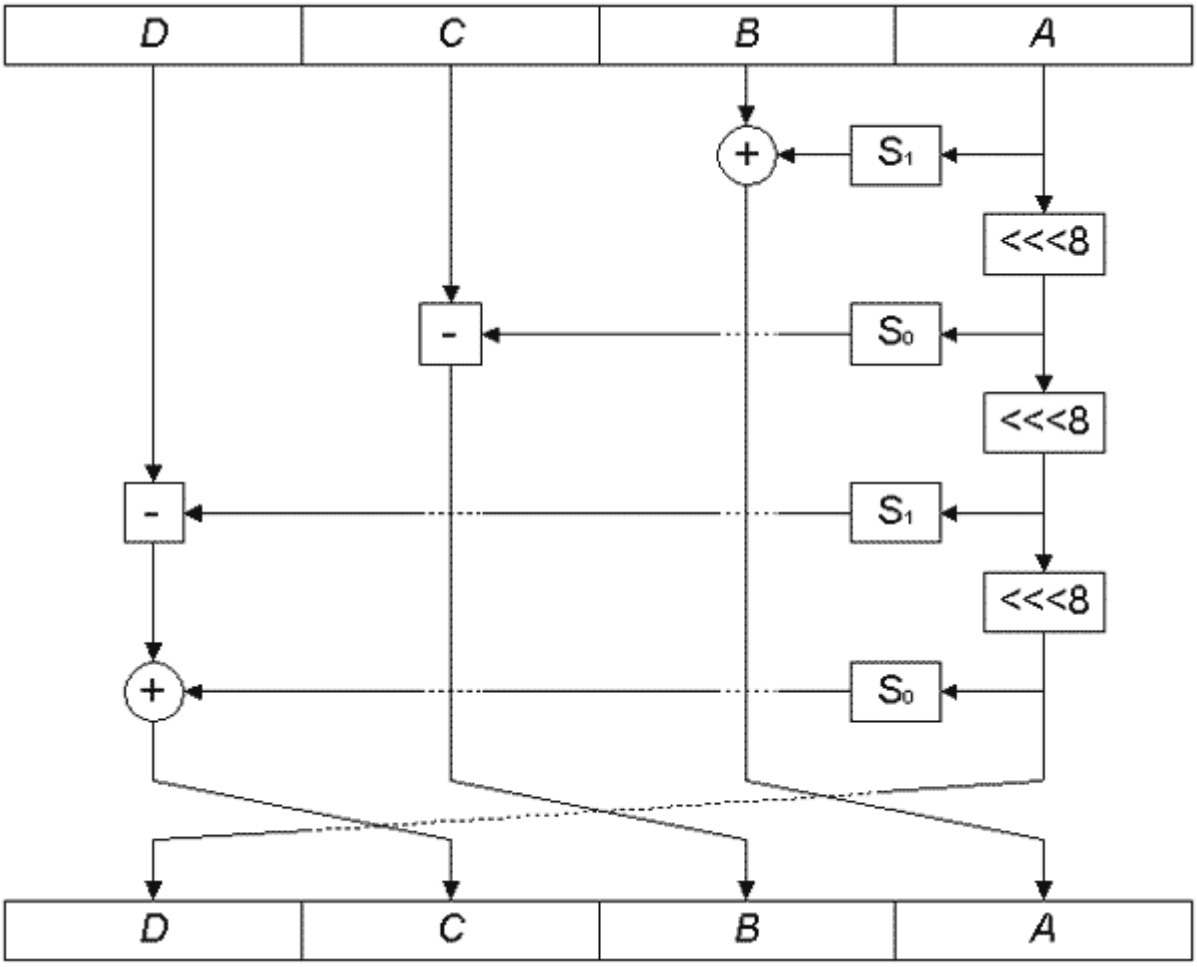


Рисунок 1.19 - Раунд обратного перемешивания алгоритма MARS.

Раунд обратного перемешивания, на рисунке 1.19 более существенно отличается от прямого. Фактически, обратное перемешивание выполняет обратные операции в обратной последовательности , смотреть рисунок 1.15 и 1.19):

1. Значение субблока **A** прогоняется через таблицу замен $\mathbf{S_1}$ и накладывается на субблок **B**операцией XOR.
2. Исходное значение субблока **A** вращается на 8 бит влево.
3. Результат предыдущего шага обрабатывается таблицей замен $\mathbf{S_0}$ и накладывается на субблок **C** операцией вычитания по модулю 2^{32}.
4. Результат шага 2 вращается на 8 бит влево.
5. Результат предыдущего шага обрабатывается таблицей замен $\mathbf{S_1}$ и накладывается на субблок **D** операцией вычитания по модулю 2^{32}.
6. Результат шага 4 вращается на 8 бит влево.
7. Результат предыдущего шага обрабатывается таблицей замен $\mathbf{S_0}$ и накладывается на субблок **D** операцией XOR.
8. Субблоки меняются местами, как показано на рис. 1.19.

Аналогично прямому перемешиванию, в некоторых раундах выполняются следующие дополнительные операции, не показанные на рис. 1.19:

1. В раундах 1 и 5 после шага 7 выполняется наложение значения субблока **A** на субблок **B**операцией вычитания по модулю 2^{32}.
2. В раундах 2 и 6 субблок C аналогичным образом накладывается на субблок **B**.

Расшифрование выполняется применением обратных операций в обратной последовательности; подробно расшифрование описано в спецификации алгоритма.

1.4.2 Процедура расширения ключа

Ключ шифрования алгоритма MARS может иметь любой размер в диапазоне от 128 до 448 бит включительно, кратный 32 битам. Расширение ключа представляет собой формирование на основе ключа шифрования 40 подключей по 32 бита каждый (причем, подключи должны обладать определенными характеристиками, о которых будет сказано ниже), для чего выполняются следующие операции:

1. Формируется временный массив $\mathbf{T_0...T_{14}}$:

$$\mathbf{T_0 = KI_0,}$$
$$\mathbf{T_1 = KI_1,}$$
$$\mathbf{...}$$
$$\mathbf{T_{n-1} = KI_{n-1},}$$
$$\mathbf{T_n = n,}$$
$$\mathbf{T_{n+1} = T_{n+2} = ... = T_{14} = 0,}$$

где $\mathbf{KI_0...KI_{n-1}}$ — исходный ключ шифрования,
n — его размер в 32-битных словах — от 4 до 14 включительно.

2. Данный шаг повторяется 4 раза (каждая итерация дает 10 вычисленных фрагментов расширенного ключа) и содержит следующие операции:

линейное преобразование:

$$\mathbf{T_i = T_i \text{ Å } ((T_{i-7 \bmod 15} \text{ Å } T_{i-2 \bmod 15}) <<< 3) \text{ Å } (4i + j),}$$

где **j** — номер итерации, начиная с 0,
i = 0...14;

- перемешивание массива **T**:

$$\mathbf{T_i = (T_i \text{ Å } S(T_{i-1 \bmod 15})) <<< 9,}$$

где **S** — табличная замена, выполняемая по тем же правилам, что и в операции **E**;

заключительная перестановка:

$$\mathbf{k_{10j+n} = T_{4n \bmod 15},}$$

где **n** = 0...9.

3. Наложение на вычисленные подключи дополнительных требований, состоящих в следующем:

- Каждый подключ, используемый для умножения в операции **E** (то есть подключи с нечетными индексами от $\mathbf{k_5}$ до $\mathbf{k_{35}}$ включительно), должен иметь нечетное значение. Мало того, единичными должны быть два младших бита подключа, а не один.
- Те же подключи не должны содержать 10 нулевых или 10 единичных бит подряд.

Модификация подключей в соответствии с данными требованиями выполняется следующим образом:

- 2 младших бита обрабатываемого подключа устанавливаются в единичные значения; старое значение запоминается в переменной **j** (оно будет использовано впоследствии). Результирующее значение подключа обозначается как **W**.
- Вычисляется 32-битная маска **M**, которая будет использована для модификации подключа — для обеспечения отсутствия в нем 10 подряд нулевых или единичных бит. Маска вычисляется следующим образом:

  A) Устанавливаются в 1 биты **M**, соответствующие тем битам обрабатываемого подключа, которые входят в последовательности из 10 нулевых или 10 единичных бит. Остальные биты устанавливаются в 0.

  B) Обнуляются те единичные биты маски **M**, которые соответствуют любому из условий:

$$\mathbf{i < 2,}$$
$$\mathbf{i = 31,}$$
$$\mathbf{W_i != W_{i-1},}$$
$$\mathbf{W_i != W_{i+1},}$$

где i — номер бита, начиная с 0, а W_i — значение i-го бита.

- Используется таблица корректирующих значений B, определенная следующим образом:

$$B_0 = A4A8D57B,$$
$$B_1 = 5B5D193B,$$
$$B_2 = C8A8309B,$$
$$B_3 = 73F9A978.$$

  Элементы таблицы B эквивалентны элементам с 265-го по 268-й включительно объединенной таблицы S; именно эти элементы используются для коррекции подключей благодаря их специфическим свойствам.

  С помощью данной таблицы следующим образом вычисляется финальное значение подключа K_i:

$$\mathbf{K_i = W \text{ Å } ((B_j <<< K_{i-1}) \& M),}$$

где **&** — логическая побитовая операция «и», а
вращение $\mathbf{B_j}$ определяется пятью младшими битами предыдущего подключа $\mathbf{K_{i-1}}$.

Описанная процедура гарантирует требуемые свойства у подключей.

1.5 Алгоритм шифрования RC6

Алгоритм RC6 был разработан в 1998 г. рядом специалистов научного подразделения известнейшей фирмы RSA Data Security — RSA Laboratories: Рональдом Ривестом (Ronald Rivest, основатель RSA Data Security), Мэттом Робшоу (Matt Robshaw), Рэем Сидни (Ray Sidney) и Икван Лайзой Ин (Yiqun Lisa Yin) специально для участия в конкурсе AES. Алгоритм во многом унаследовал черты предыдущего алгоритма Рональда Ривеста — 64-битного блочного шифра RC5, разработанного в 1997 г. Фактически, алгоритм претерпел два принципиальных изменения:

- в отличие от RC5, в алгоритме используется умножение по модулю 2^{32};
- для сохранения 32-битных вычислений вместо разбиения шифруемого блока данных

1.5.1 Структура алгоритма

Как и алгоритм RC5, RC6 имеет гибкую структуру: помимо секретного ключа, параметрами алгоритма являются следующее:

- размер слова **w**; RC6 шифрует блоками по 4 слова;
- количество раундов алгоритма **R**;
- размер секретного ключа в байтах **b**.

Аналогично RC5, для уточнения параметров алгоритма, используемых в его конкретной реализации, применяется обозначение RC6-**w/R/b**. Поскольку в конкурсе AES 128-битный блок является обязательным, значение w фиксировано и равно 32. В спецификации алгоритма фиксируется также количество раундов: **R** = 20. Поскольку конкурс AES предусматривает использование трех фиксированных размеров ключей, рассмотрим три следующих варианта алгоритма: RC6-32/20/16, RC6-32/20/24 и RC6-32/20/32, совокупность которых и будет подразумеваться далее под названием «RC6». Структура алгоритма представлена на рисунке 1.20.

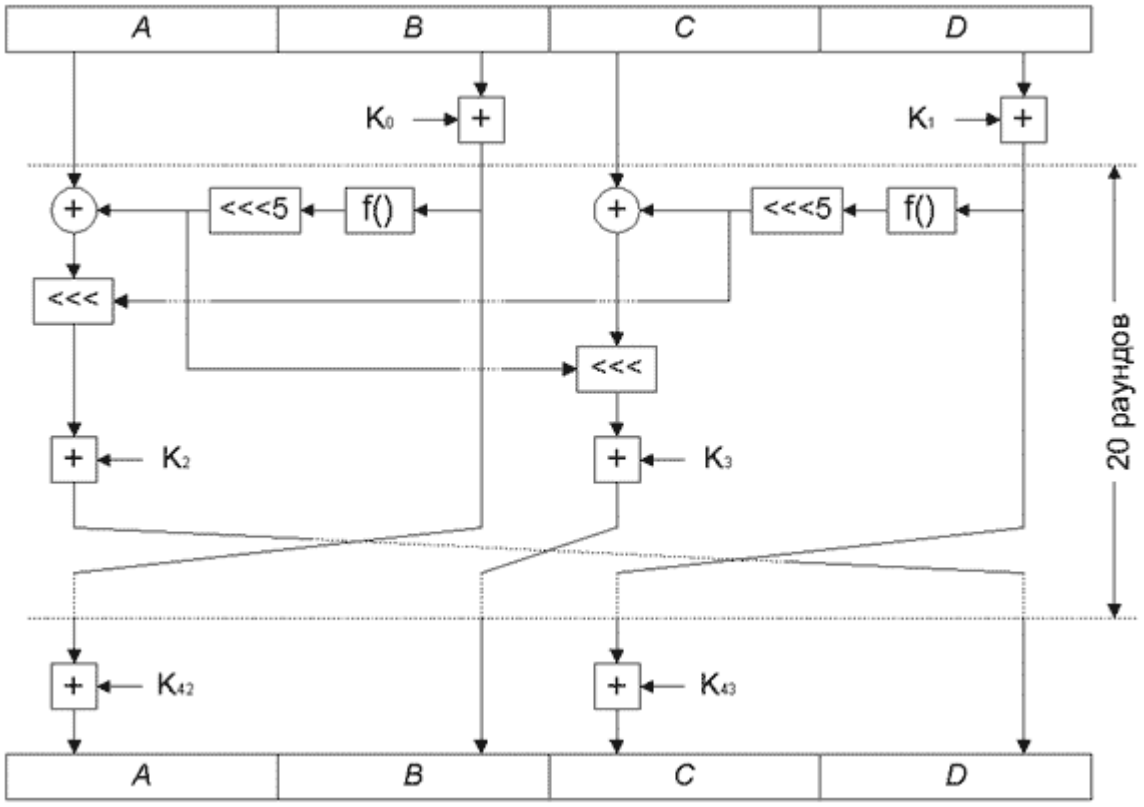


Рисунок 1.20 - Структура алгоритма RC6.

Как было сказано выше, в алгоритме используется 20 раундов преобразований, перед которыми выполняется частичное входное отбеливание:

$$B = B + K_0 \bmod 2^{32},$$
$$D = D + K_1 \bmod 2^{32},$$

где **A**, **B**, **C**, **D** — текущие значения обрабатываемых 32-битных субблоков, $K_0 ... K_{43}$ — фрагменты расширенного ключа.

Аналогичным образом выполняется частичное выходное отбеливание:

$$A = A + K_{42} \bmod 2^{32},$$
$$C = C + K_{43} \bmod 2^{32}.$$

В каждом раунде алгоритма выполняются следующие действия:

$$t_1 = f(B) <<< 5,$$
$$t_2 = f(D) <<< 5,$$
$$A = ((A \text{ Å } t_1) <<< t_2) + K_{2i} \bmod 2^{32},$$
$$C = ((C \text{ Å } t_2) <<< t_1) + K_{2i+1} \bmod 2^{32},$$

где t_1 и t_2 — временные переменные,

количество бит вращения на переменное число бит определяется значением 5 младших бит параметра (t_1 или t_2),

функция **f()** выполняет следующее квадратичное преобразование:

$$f(x) = x * (2x + 1) \bmod 2^{32}.$$

В конце каждого раунда выполняется сдвиг субблоков — смотреть рисунок 1.20.

При расшифровании подключи используются в обратном порядке, наложение подключей вместо сложения по модулю 2^{32} выполняется вычитанием, а также сдвиг субблоков выполняется в начале раунда и в обратную сторону. Преобразование **f()** не претерпело изменений, на рисунке 1.21.

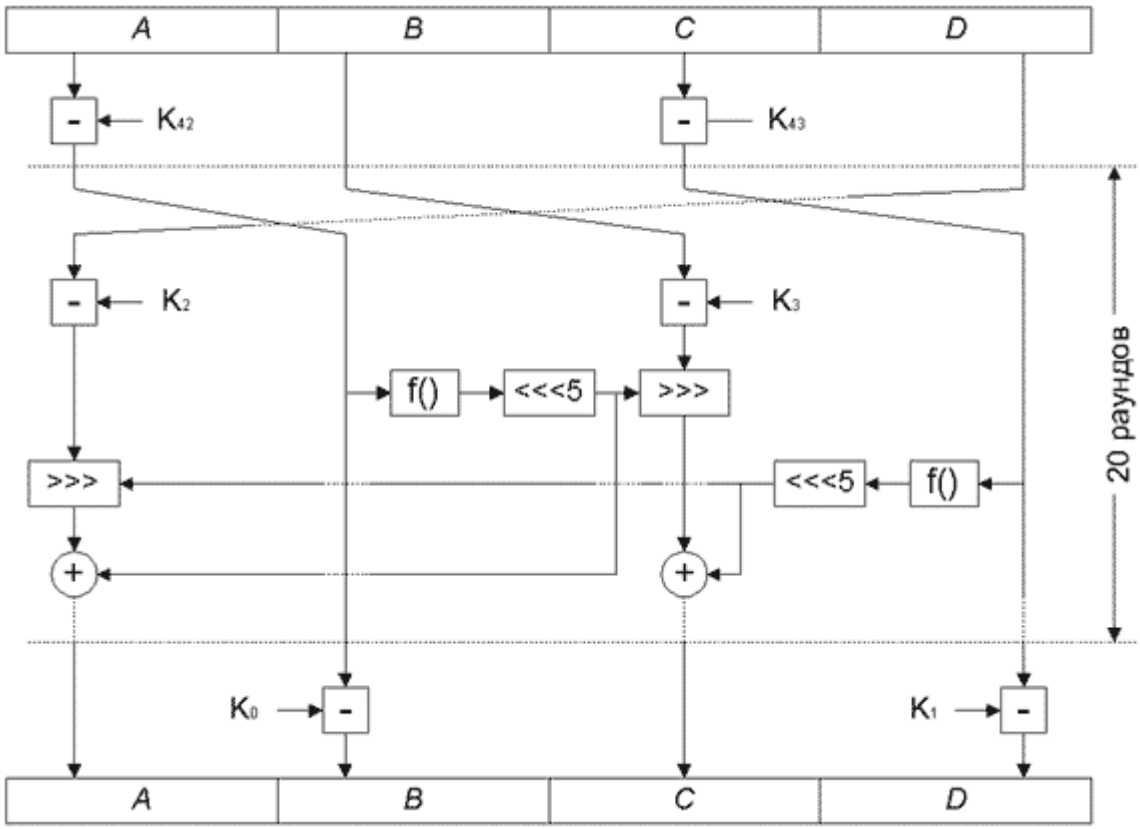


Рисунок 1.21 - Расшифрование алгоритмом RC6.

1.5.2 Процедура расширения ключа

Процедура расширения ключа RC6 аналогична RC5, за исключением того, что RC6 требуется существенно больше сгенерированных подключей, а именно 2**R**+4, то есть $\mathbf{K_0...K_{43}}$ для 20 раундов. Рассмотрим данную процедуру для алгоритма RC6 в варианте для конкурса AES, то есть с указанными выше фиксированными параметрами.
Расширение ключа выполняется в два этапа:

1. Инициализация массива расширенных ключей $\mathbf{K_0...K_{43}}$, которая производится следующим образом:

$$\mathbf{K_0 = P_{32}},$$
$$\mathbf{K_{i+1} = K_i + Q_{32} \bmod 2^{32}},$$

где $\mathbf{P_{32}}$ и $\mathbf{Q_{32}}$ — псевдослучайные константы, образованные путем умножения на 2^{32}дробной части и последующего округления до ближайшего нечетного целого двух математических констант (**e** и соответственно):

$$P_{32} = B7E15163,$$
$$Q_{32} = 9E3779B9,$$

Авторы алгоритма в его спецификации [14] утверждают, что выбор данных значений не является важным. Соответственно, аналогично описанному выше алгоритму Twofish-FK, при необходимости создания реализации алгоритма RC6, не совместимой со стандартной, следует изменить значения P_{32} и Q_{32}.

2. Циклически выполняются следующие действия:

$$A = K_i = (K_i + A + B \bmod 2^{32}) <<< 3,$$
$$B = KI_j = (KI_j + A + B \bmod 2^{32}) <<< (A + B),$$
$$i = i + 1 \bmod 44,$$
$$j = j + 1 \bmod c,$$

где i, j, A и B — временные переменные, их начальные значения равны нулю, KI — исходный ключ шифрования, представленный в виде с 32-битных слов. Выполняется 3c итераций цикла.

1.6. Описание показателей алгоритмов шифрования

Результаты проделанной работы по изучению алгоритмов-финалистов NIST сформулировал в виде отчета. Данный отчет содержит как результаты анализа алгоритмов, так и обоснование критериев, по которым выполнялась оценка.

На основе отчета кратко сформулировал сравнительные оценки пяти алгоритмов-финалистов конкурса AES по основным критериям в виде следующей таблицы:

№	Категория	Serpent	Twofish	MARS	RC6	Rijndael
1	Криптостойкость	+	+	+	+	+
2	Запас криптостойкости	++	++	++	+	+
3	Скорость шифрования при программной реализации	-	±	±	+	+
4	Скорость расширения ключа при программной реализации	±	-	±	±	+
5	Смарт-карты с большим объемом ресурсов	+	+	-	±	++
6	Смарт-карты с ограниченным объемом ресурсов	±	+	-	±	++
7	Аппаратная реализация (ПЛИС)	+	+	-	±	+
8	Аппаратная реализация (специализированная микросхема)	+	±	-	-	+
9	Защита от атак по времени выполнения и потребляемой	+	±	-	-	+

	мощности[3]					
10	Защита от атак по потребляемой мощности на процедуру расширения ключа	±	±	±	±	-
11	Защита от атак по потребляемой мощности на реализации в смарт-картах	±	+	-	±	+
12	Возможность расширения ключа «на лету»	+	+	±	±	±
13	Наличие вариантов реализации (без потерь в совместимости)	+	+	±	±	+
14	Возможность параллельных вычислений	±	±	±	±	+

1.7 Влияние алгоритмов шифрования на качество информационных систем

1. Криптостойкость всех алгоритмов-финалистов оказалась достаточной — в процессе исследований не было обнаружено каких-либо реально реализуемых атак на полноценные и полнораундовые версии алгоритмов. В данном случае криптоаналитики обычно исследуют варианты алгоритмов с усеченным числом раундов, либо с некоторыми внесенными изменениями, незначительными, но ослабляющими алгоритм. Под запасом криптостойкости (security margin) эксперты NIST подразумевают соотношение полного (предусмотренного в спецификациях алгоритмов) числа раундов и максимального из тех вариантов, против которых действуют какие-либо криптоаналитические атаки. Например, с помощью дифференциально-линейного криптоанализа вскрывается 11-раундовый Serpent , тогда как в оригинальном алгоритме выполняется 32 раунда. Эксперты NIST в отчете предупредили, что данная оценка является весьма поверхностной и не может быть значимой при выборе алгоритма-победителя конкурса, но, тем не менее, отметили, что запас криптостойкости у Rijndael и RC6 несколько ниже, чем у остальных алгоритмов-финалистов.

2. В пп. 5-8 приведена сравнительная оценка возможности и эффективности реализации алгоритмов в перечисленных устройствах.

3. В пп. 9-11 имеется в виду, насколько операции, выполняемые конкретным алгоритмом, могут быть подвержены анализу указанным методом. При этом принималось в расчет то, что операции могут быть

модифицированы с целью усложнения криптоанализа за счет потери в скорости шифрования (например, проблемное в этом смысле вращение на переменное число бит может принудительно выполняться за равное число тактов — то есть максимально возможное для данной операции; именно подобные меры противодействия атакам по времени исполнения и потребляемой мощности рекомендует их изобретатель Пол Кохер (Paul C. Kocher).

4. Из описаний алгоритмов видно, что все они поддерживают расширение ключа «на лету» (то есть подключи могут генерироваться непосредственно в процессе шифрования — по мере необходимости), однако, только Serpent и Twofish поддерживают такую возможность без каких-либо ограничений

5. Под *наличием вариантов реализации* (implementation flexibility) имеется в виду возможность различным образом реализовывать какие-либо операции алгоритма с оптимизацией под конкретные цели. Наиболее показательными в этом смысле являются упомянутые ранее варианты процедуры расширения ключа алгоритма Twofish , позволяющие оптимизировать реализацию алгоритма в зависимости, прежде всего, от частоты смены ключа.

Сформулируем основные достоинства и недостатки каждого из рассмотренных в данной статье алгоритмов-финалистов :

Алгоритм	**Достоинства**	**Недостатки**
Serpent	1. Простая структура алгоритма облегчает его анализ с целью нахождения возможных уязвимостей. 2. Serpent эффективно реализуем аппаратно и в условиях ограниченных ресурсов. 3. Serpent легко модифицируется с целью защиты от атак по времени выполнения и потребляемой мощности (однако, за счет снижения скорости).	1. Является самым медленным из алгоритмов-финалистов в программных реализациях. 2. Процедуры зашифрования и расшифрования абсолютно различны, т.е. требуют раздельной реализации. 3. Распараллеливание вычислений при шифровании алгоритмом Serpent реализуемо с ограничениями.
Twofish	1. Twofish эффективно	1. Сложность структуры

	реализуем аппаратно и в условиях ограниченных ресурсов. 2. Зашифрование и расшифрование в алгоритме Twofish практически идентичны. 3. Является лучшим из алгоритмов-финалистов с точки зрения поддержки расширения ключа «на лету». 4. Несколько вариантов реализации позволяют оптимизировать алгоритм для конкретных применений.	алгоритма затрудняет его анализ. 5. Сложная и медленная процедура расширения ключа. 6. Относительно сложно защищается от атак по времени выполнения и потребляемой мощности. 7. Распараллеливание вычислений при шифровании алгоритмом Twofish реализуемо с ограничениями.
MARS	Зашифрование и расшифрование в алгоритме MARS практически идентичны.	1. Исключительно сложная структура алгоритма с раундами различных типов затрудняет как анализ алгоритма, так и его реализацию. 2. Возникают проблемы при программной реализации на тех платформах, которые не поддерживают 32-битное умножение и вращение на переменное число бит. 3. Алгоритм MARS не может быть эффективно реализован аппаратно и в условиях ограниченных ресурсов. 4. Сложно защищается от атак по времени выполнения и потребляемой мощности. 5. MARS хуже других алгоритмов-финалистов поддерживает расширение ключей «на лету». 6. Распараллеливание вычислений при шифровании алгоритмом

		MARS реализуемо с ограничениями.
RC6	1. Простая структура алгоритма облегчает его анализ. Кроме того, алгоритм унаследовал часть преобразований от своего предшественника — алгоритма RC5, тщательно проанализированного до конкурса AES. 2. Самый быстрый из алгоритмов-финалистов на 32-битных платформах. 3. Зашифрование и расшифрование в алгоритме RC6 практически идентичны.	1. Скорость шифрования при программной реализации сильно зависит от того, поддерживает ли платформа 32-битное умножение и вращение на переменное число бит. 2. RC6 сложно реализуем аппаратно и в условиях ограниченных ресурсов. 3. Достаточно сложно защищается от атак по времени выполнения и потребляемой мощности. 4. Недостаточно полно поддерживает расширение ключей «на лету». 5. Распараллеливание вычислений при шифровании алгоритмом RC6 реализуемо с ограничениями.

2.Парольная система авторизации с применением технологии шифрования методом “нелинейной динамики” Captcha

2.1 Аутентификация как базис безопасности

2.1.1 Этапы входа в систему

Современные веб-ресурсы содержат в себе множество функционала, который по разному доступен различным пользователям. Основной вопрос заключается в следующем: как обеспечить, чтобы система узнавала пользователей и в соответствии с результатом распознавания – предоставляла им тот или иной функционал? Безопасность процедуры входа для пользователей во многом определяет защищенность информационной системы в целом.

Процедуру входа зачастую именуют по-разному: войти, авторизоваться, аутентифицироваться, залогиниться. Обычно все понимают, о чем идет речь, но методологически термины используются не всегда корректно...

В теории информационной безопасности процесс входа в систему делится на 3 основные стадии:

1. Идентификация (Identification)
   Под идентификацией, применительно к обеспечению информационной безопасности компьютерной системы, понимают однозначное распознавание уникального имени субъекта (пользователя).
2. Аутентификация (Authentication)
   Аутентификация обеспечивает подтверждение подлинности субъекта, то есть подтверждение того, что предъявленное имя соответствует данному субъекту.
3. Авторизация (Authorization)
   На третьем шаге, система проводит процедуру авторизации, когда на основании результата процедуры аутентификации она дает пользователю тот или иной уровень доступа.

Этап аутентификации является самым важным в этой цепочке и основная проблема заключается в том, чтобы сделать эту процедуру максимально точной и безопасной.

Аутентификация бывает односторонней (обычно только клиент доказывает свою подлинность серверу) и двусторонней (взаимной). Пример односторонней аутентификации - процедура входа пользователя в систему, например в продукты «1С-Битрикс»

2.1.2 Способы аутентификации

Способы аутентификации пользователей в компьютерных системах делят на три группы:

- К первой группе относятся способы аутентификации, основанные на том, что пользователь знает некоторую подтверждающую его подлинность информацию (обычно это парольная аутентификация).
- Ко второй группе относятся способы аутентификации, основанные на том, что пользователь имеет некоторый материальный объект, который может подтвердить его подлинность (например, пластиковую карту с идентифицирующей пользователя информацией).
- К третьей группе относятся способы аутентификации, основанные на таких данных, которые позволяют однозначно считать, что пользователь и есть тот самый субъект, за которого себя выдает (биометрические данные, особенности клавиатурного почерка и т.п.).

2.2 Парольная аутентификация

Использование пароля в качестве аутентификационного фактора, наверное, еще очень долго будет являться наиболее распространенным способом решения задач определения подлинности. В первую очередь, в силу своей простоты и низких затрат на обеспечение всей инфраструктуры, особенно если речь идет о веб-приложениях.

Парольная защита заслуженно считается не очень надежной. В программных продуктах мы постарались максимально возможно усилить аутентификацию пользователей с использованием паролей. В следующих пунктах показано за счет чего это достигается.

2.2.1 Задание стойкости паролей

При выборе паролей пользователи системы (например, веб-сайта) должны руководствоваться двумя, по сути взаимоисключающими, правилами - пароли должны трудно подбираться и легко запоминаться (поскольку пароль ни при каких условиях не должен нигде записываться, так как в этом случае необходимо будет дополнительно решать задачу защиты носителя пароля).

Но поскольку правила взаимоисключающие, пароли обычно простые и очень хорошо подбираются, а также записываются повсеместно без надлежащей защиты этих самых записей. Если вторую неприятность решить технически невозможно, да и организационные меры особо не помогают, то вот с обеспечением сложности пароля можно поработать.

Сложность подбора пароля определяется, в первую очередь, мощностью множества символов, используемого при выборе пароля (N), и минимально возможной длиной пароля (к). В этом случае число различных паролей

может быть оценено снизу как, $C_p = N^k$. Например, если множество символов пароля образуют строчные латинские буквы, а минимальная длина пароля равна 3, то $C_p = 26^3 = 17576$ (что совсем немного для программного подбора). Если же множество символов пароля состоит из строчных и прописных латинских букв, а также из цифр и минимальная длина пароля равна 6, то $C_p = 62^6 = 56800235584$.

Сложность выбираемых пользователями КС паролей должна устанавливаться администратором при реализации установленной для данной системы политики безопасности.

Другими параметрами политики учетных записей при использовании парольной аутентификации являются:

- максимальный срок действия пароля (любой секрет не может сохраняться в тайне вечно);
- несовпадение пароля с логическим именем пользователя, под которым он зарегистрирован в системе;
- неповторяемость паролей одного пользователя.

С точки зрения теории, эти рекомендации очень полезны и обычно применяются в совокупности, но как правило на практике они не дают сколько-либо весомого усиления защиты. Так, например, требование неповторяемости паролей может быть реализовано двумя способами. Во-первых, можно установить минимальный срок действия пароля (в противном случае пользователь, вынужденный после истечения срока действия своего пароля поменять его, сможет тут же сменить пароль на старый). Во-вторых, можно вести список уже использовавшихся данным пользователем паролей (максимальная длина списка при этом может устанавливаться администратором).

К сожалению, обеспечить реальную уникальность каждого вновь выбираемого пользователем пароля с помощью приведенных выше мер практически невозможно. Пользователь может, не нарушая установленных ограничений, выбирать пароли «A1», «A2», ... где A — первый пароль пользователя, удовлетворяющий требованиям сложности. Тоже самое касается несовпадения логина с паролем. Ничто не помешает сделать логин 'dima', а пароль 'dima_1'.

2.3 Противодействие подбору паролей

Настраивая политику учетных записей пользователей необходимо назначить меры противодействия системы попыткам подбора паролей. Для этого могут применяться следующие правила:

1. Препятствие подбору пароля автоматизированным способом.
2. Ограничение числа попыток входа в систему
3. Скрытие логического имени последнего работавшего пользователя
4. Учет всех попыток (успешных и неудачных) входа в систему

Для воспрепятствования автоматизированным способам подбора паролей для современных веб-приложении мы будем использовать CAPTCHA (Completely Automatic Public Turing Test to Tell Computers and Humans Apart - полностью автоматический тест Тьюринга для различения компьютеров и людей). CAPTCHA представляет собой задачу, сложную для решения компьютером и простую для человека, и обычно реализуется в виде изображения, на котором необходимо различить набор символов или другую сигнатуру.

Применяются CAPTCHA для того, чтобы предотвратить множественные автоматические регистрации и отправления сообщений программами-роботами. Т. е. задача CAPTCHA — защита от спама, флуда и захвата аккаунтов.

Чаще всего CAPTCHA выглядит как тем или иным образом зашумленное случайное число, слово или иная надпись, которую пользователю нужно прочитать и ввести прочитанный результат, хотя существуют и другие алгоритмы. Примеры CAPTCHA показаны на рисунке 2.1.

Рисунок 2.1 - CAPTCHA

2.4 Новый метод создания паролей с помощью CAPTCHA

Совместная группа разработчиков из Физического института имени Макса Планка в Германии и компании Axioma Research предложили новый метод создания паролей, которые с одной стороны трудно взломать, а с другой - легко запомнить.

Исследователи говорят, что они объединили так называемую "нелинейную динамику" и хаос для создания зашифрованной системы p-captcha (p от Password).

Суть новинки заключается в следующем: пользователь запоминает лишь часть пароля, а специальный Java-апплет "вспоминает" остальную часть.

Например, у пользователя есть важное приложение или файл, которые необходимо зашифровать. Вначале пароль бьется на две части, условно говоря, простую и сложную. Пользователь запоминает простую, а Java-апплет создает captcha для сложной. Когда p-captcha зашифровывается, пользователю нужно лишь ввести простую часть, а затем открыть captcha, что позволит полностью дешифровать информацию, защищенную новой системой.

Авторы системы говорят, что систему captcha можно сколько угодно усложнять, например, создавая трехмерные изображения, увеличивая длину символов и другими способами. По словам разработчиков, против новой системы становится бесполезен метод перебора паролей, а также различные методы перехвата данных, так как captcha может генерироваться динамически каждый раз.

Программисты говорят, что новый метод защиты целесообразно внедрять не только в шифровании данных, но и на веб-сервисах, например в социальных сетях и системах электронной почты .

2.5 Модифицированный алгоритм шифрование двухфакторной аутентификации с применением Captcha

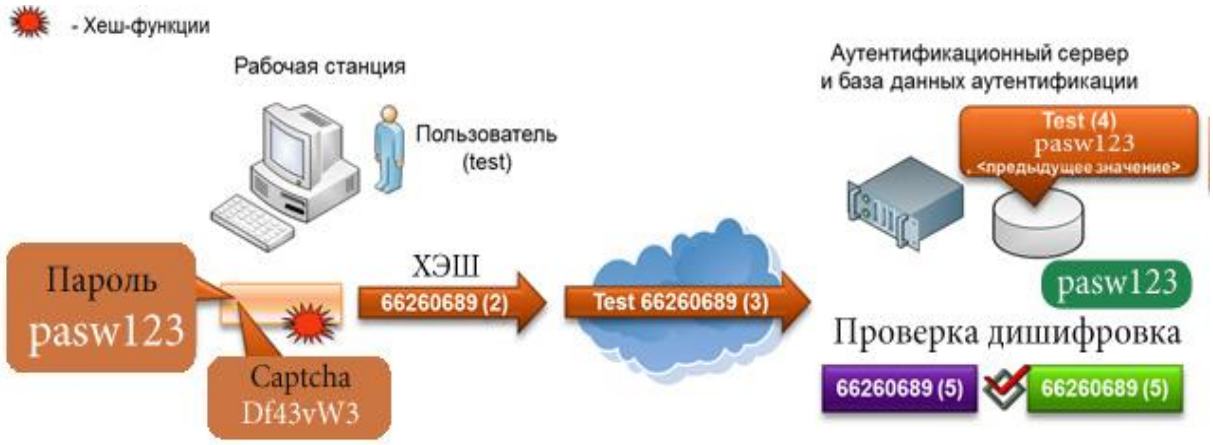
Привычный, ставший стандартным метод – использование долговременных, запоминаемых паролей. Этот способ проверки «правильности» пользователя в рамках современных ИТ-решений должен уйти в прошлое, хотя и имеет место в огромном числе компаний, в силу простоты использования и реализации.

Тем не менее использование такой аутентификации скорее должно свидетельствовать о незрелости ИТ-инфраструктуры с точки зрения безопасности или об отсутствии потребности бизнеса в построении защищенной информационной системы. Парольная аутентификация потенциально уязвима ввиду использования социотехники, внедрения клавиатурных шпионов, возможности перехвата и т.п.

Аутентификация – процедура проверки подлинности, позволяющая достоверно убедиться в том, что, предъявивший свой идентификатор на самом деле является именно тем, за кого он себя выдает. Для этого он должен подтвердить факт обладания информацией, которая может быть доступна только ему одному (пароль, ключ и т.п.).

На данной схеме присутствует блок процедуры установления подлинности p-captcha.

Принципы работы



1. Пользователь вводит свое имя, секретный ключ и captha на рабочей станции.
2. Captha с ключом перемешиваются в открытом виде преобразуя шифр.
3. Этот шифр дополнительно шифруется алгоритмом хэширования MD5
4. Хэш передается по сети серверу для расшифровки.

5. Сервер расшифровывает хэш
6. После расшифровки сравниваются два значения captha: полученное от пользователя и вычисленное сервером.
7. При их совпадении аутентификация считается успешной.
8. Сервер осуществляет поиск записи пользователя в аутентификационной БД, используя секретный ключ пользователя, и к полученному ключу сервер подставляет вторую половину пароля.

Пошаговый процесс авторизации с применением Captcha

- **Шаг 1.** Шифрование начинается в окне авторизации пользователя. В этой форме добавлена форма заполнения кода captcha. На первый взгляд пользователь не о чем не подозревает. Пользователь вводит только первую часть пароля который с капчай хаотично перемешается.
- **Шаг 2.** Пароль перемешанный с капчай хэшируеться 128-битныйм алгоритмом хеширования MD5. (можно использовать любой алгоритм хэширования мы выбрали MD5). Для сведения алгоритмы хеширования предназначены для создания «отпечатков» или дайджестов сообщения произвольной длины и последующей проверки их подлинности.

- **Шаг 3.** Зашифрованный хэш отправляется на сервер где происходит дешифровка. Дешифровка идет в обратном порядке что шифровка. После получения первая часть пароля соединяется со второй частью пароля который храниться на сервере.

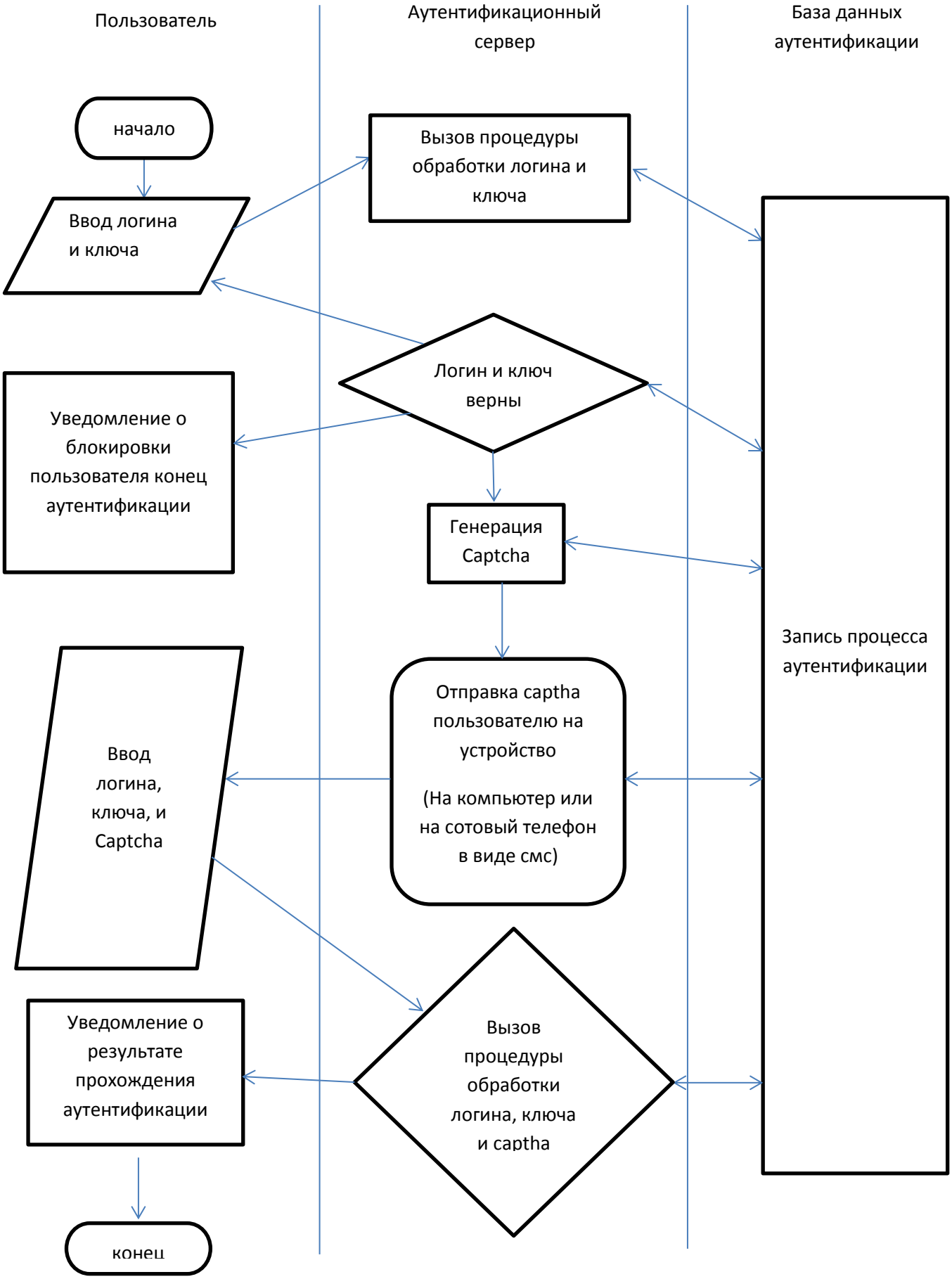
Двухфакторная аутентификация

Двухфакторная аутентификация обеспечивает дополнительную защиту, при входе в систему запрашиваются не только имя пользователя и пароль, но и уникальные "коды подтверждения". Даже если злоумышленник узнает ваш пароль, получить доступ к аккаунту ему не удастся.

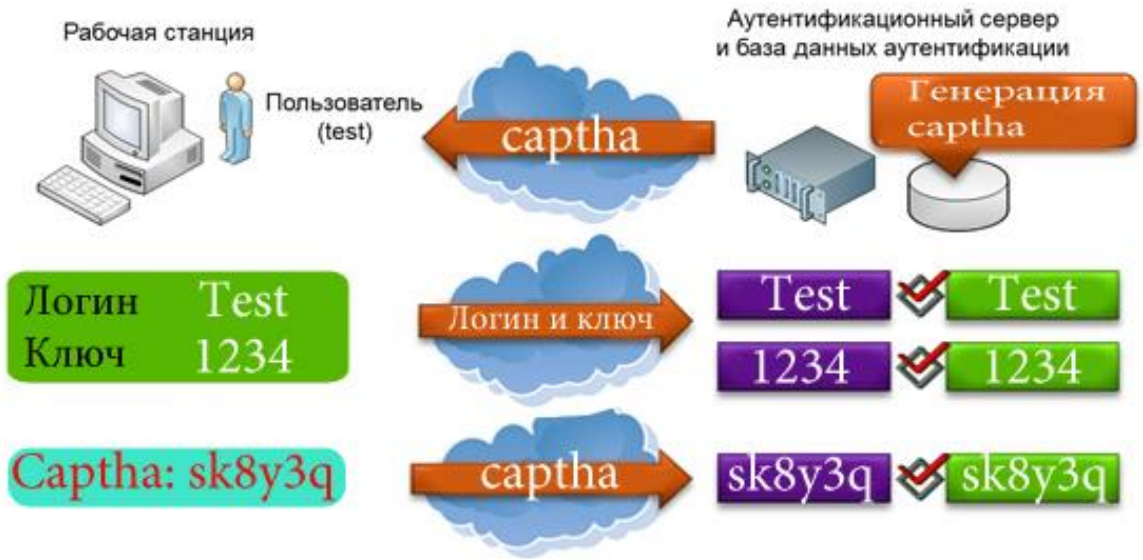
С технической точки зрения применение двухфакторной аутентификации позволит минимизировать риски, связанные с использованием долговременных паролей.

Взяв в основу идею новой парольной защиты «p-CAPTCHA» мы доработали её с применением двухфакторной аутентификации.

Блок схема двухфакторной аутентификации



Принципы работы двухфакторной аутентификации с применением captha



1. Пользователь вводит свой логин и ключ на рабочей станции. Имя пользователя и ключ передается серверу по сети в открытом виде.
2. Сервер проводит авторизацию пользователя логина и ключа. Обрабатывает авторизацию в базе данных пользователей.
3. При успешной авторизации в базе данных, сервер генерирует определенный случайный запрос в виде captha на этот ключ пользователя.
4. Сервер шифрует запрос с помощью секретного ключа пользователя, и формируется ответное значение, которое передается по сети в открытом виде пользователю на рабочую станцию в виде captha.
5. Пользователь вводит полученные данные (captha) на своей рабочей станции. Captha передается серверу по сети в открытом виде.
6. Сервер осуществляет поиск записи пользователя в аутентификационной БД, используя секретный ключ и полученную captha пользователя.
7. Далее сравниваются два значения: полученное от пользователя и вычисленное сервером. При их совпадении аутентификация считается успешной.

Данный вид шифрования двухфакторной авторизации не требует запоминания больших и сложных паролей. Ключ может быть в виде четырехзначного пин кода. Шифрование проходит в открытом виде, что не требует дополнительных вычислительных средств. Попытка ввода ключа осуществляется максимально сокращенно (2-3 раза) и в определенный момент времени (в течении 30-60 секунд). В случаи неправильного ввода

ключа, пользователь блокируется и получает уведомление о попытке взлома. Каждая авторизация записывается в базу данных сервера авторизации. Случайный запрос в виде капчи может передаваться пользователю, как на рабочую станцию так и на мобильное устройство в виде смс.

3.Исследование влияния алгоритмов шифрования на качество информационных систем

На сегодняшний день существует огромное множество различных алгоритмов и средств защиты информации. Одним из таких способ является криптография с использованием алгоритма MD5.

MD5 (Message Digest 5) один из серии алгоритмов по построению дайджеста сообщения, разработанный профессором Рональдом Л. Ривестом из Массачусетского технологического института. Разработан в 1991 году, как более надёжный вариант предыдущего алгоритма MD4. Используется для проверки подлинности опубликованных сообщений путём сравнения дайджеста сообщения с опубликованным.

Процесс шифрования

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения может быть любой (в том числе нулевой). Запишем длину сообщения в L. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

Шаг 1. Выравнивание потока.

Сначала дописывают единичный бит в конец потока (байт 0x80), затем необходимое число нулевых бит. Входные данные выравниваются так, чтобы их новый размер L' был сравним с 448 по модулю 512 (L’ = 512 × N + 448). Выравнивание происходит, даже если длина уже сравнима с 448.

Шаг 2. Добавление длины сообщения.

В оставшиеся 64 бита дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта. Если длина превосходит $2^{64} - 1$, то дописывают только младшие биты. После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

Шаг 3. Инициализация буфера.

Для вычислений инициализируются 4 переменных размером по 32 бита и задаются начальные значения шестнадцатеричными числами (шестнадцатеричное представление, сначала младший байт):

A = 01 23 45 67;
B = 89 AB CD EF;
C = FE DC BA 98;
D = 76 54 32 10.

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние **ABCD** называется инициализирующим вектором.

Определим ещё функции и константы, которые нам понадобятся для вычислений.

Потребуются 4 функции (1)-(4) для четырёх раундов. Введём функции от трёх параметров - слов, результатом также будет слово.

1 раунд $FunF(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$ (1)

2 раунд $FunG(X,Y,Z) = (X \wedge Z) \vee (\neg Z \wedge Y)$ (2)

3 раунд $FunH(X,Y,Z) = X \oplus Y \oplus Z$ (3)

4 раунд $FunI(X,Y,Z) = Y \oplus (\neg Z \vee X)$ (4)

Определим таблицу констант $T[1..64]$ - 64-элементная таблица данных, построенная следующим образом:

$$T[i] = int(4294967296 \cdot |\sin(i)|), \text{ где } 4294967296 = 2^{32} \quad (5)$$

Выровненные данные разбиваются на блоки (слова) по 32 бита, и каждый блок проходит 4 раунда из 16 операторов. Все операторы однотипны и имеют вид $[abcd\ k\ s\ i]$, определяемый по формуле (6):

$$a = b + ((a + Fun(b,c,d) + X[k] + T[i]) <<< s), \quad (6)$$

где X – блок данных;

$X[k] = M[n \cdot 16 + k]$, где k – номер 32-битного слова из n-го 512 битного блока сообщения;

s – циклический сдвиг влево на s бит полученного 32-битного аргумента.

Шаг 4. Вычисление в цикле.

Заносим в блок данных элемент n из массива. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

AA = A
BB = B
CC = C
DD = D

Суммируем с результатом предыдущего цикла:
A = AA + A
B = BB + B
C = CC + C
D = DD + D

Подробно четыре раунда преобразований показаны на рисунке 1 Приложения А.

После окончания цикла необходимо проверить, есть ли ещё блоки для вычислений. Если да, то изменяем номер элемента массива (n++) и переходим в начало цикла /2/.

3.1Криптоанализ MD5

На данный момент существуют несколько видов «взлома» хешей MD5 – подбора сообщения с заданным хешем:

- Перебор по словарю;
- Brute-force;
- Rainbow table.

Перебор, или атака по словарю – метод преодоления криптографической защиты путём перебора большого числа вариантов, однако, проверяются не все возможные варианты, а лишь уже отобранные до этого и загружаемые из списка слов, или словаря. Этот метод оказывается достаточно эффективным потому, что многие люди выбирают в качестве пароля одиночные слова или их простые вариации (например, добавляют к ним одну цифру).

Brute force (полный перебор или метод «грубой силы») – метод решения задачи путем перебора всех возможных вариантов. Сложность полного перебора зависит от количества всех возможных решений задачи. Если пространство решений очень велико, то полный перебор может не дать результатов в течение нескольких лет или даже столетий.

В криптографии на вычислительной сложности полного перебора основывается оценка криптостойкости шифров. В частности, шифр считается криптостойким, если не существует метода «взлома» существенно более быстрого чем полный перебор всех ключей. Криптографические атаки, основанные на методе полного перебора, являются самыми универсальными, но и самыми долгими.

Rainbow table (радужная таблица) – специальный вариант таблиц поиска, использующий механизм разумного компромиса между временем поиска по таблице и занимаемой памяти. Радужные таблицы используются для вскрытия паролей, преобразованных при помощи необратимой хеш-функции, а также для атак на симметричные и асимметричные шифры на основе известного открытого текста.

3.2Описание экспериментальной модели

При создании модели процесса криптоанализа, был использован язык программирования высокого уровня Visual Basic.

Данная программа позволяет произвести анализ криптостойкости различных криптографических алгоритмов, применяемых в современных информационных системах, на серверах, при хранении конфиденциальной информации (пароли, шифры и т. д.).

Интерфейс программы представляет собой окно, включающее рабочую область, панель инструментов и командную строку (рисунок 1 Приложения Б).

Процесс создания модели атаки происходит в следующем порядке:

1) Организуется модель компьютерной сети, состоящей из рабочих серверов, хранящих информацию и подвергающихся атакам (из панели инструментом путем перетаскивания в рабочую область иконок сервера и моделируемых атак).

2) В открывшихся окнах выбираются параметры каждой атаки и каждого сервера по отдельности (рисунки 1- 3 Приложения Г, Д, Е).

3) Выбираются сервера и атаки нажатием на соответствующие иконки.

4) После нажатия кнопки «Start» происходит процесс моделирования атаки на серверы, и выводятся результаты исследования для каждого сервера (рисунок 2 Приложения В).

Изменяя параметры шифрования на сервере и параметры криптоаналитических атак на серверы, можно достичь различных объемов потерянной информации.

Допустимые параметры серверов:

– тип данных: MD5;

– тип шифрования: MD5; MD5 + UNIX; MD5 + Base 64; MD5 + HMAC; MD5 + AES;

– количество символов в пароле: вводится целое число (которое ранжируется по категориям: от 0 до 3; от 4 до 8; более 8);

– типы символов в пароле: только цифры; только буквы; буквы и цифры; буквы, цифры и специальные символы.

Допустимые параметры атак:

– тип атаки: Brute; Dictionary Attack; Rainbow Attack;

– длительность атаки: вводится целое число, которое определяет длительность атаки в секундах.

На рисунке 2 Приложения Б показана модель процесса атаки, включающая 5 серверов и 5 атак. Линиями показаны направления атак на сервера.

Результаты моделирования программа выводит в виде таблицы непосредственно на рабочую область (рисунок 2 Приложения В) и дополнительно выводится в текстовой форме в файл Statistics.txt, хранящийся в корневой папке ПО.

Дополнительно программа предусматривает вызов Кодировщика MD5 – модуля, позволяющего находить MD5 любой введенной текстовой информации (рисунок 3 Приложения В).

3.3 Экспериментальная часть

Для выявления наиболее криптостойкого алгоритма была создана модель на разработанном ПО, состоящая из 5 серверов и 5 криптоаналитических атак (рисунок 1 Приложения В).

Параметры использованных серверов и атак показаны в Приложениях Г, Д, Е.

Таким образом, для каждого сервера обеспечивалось три типа атаки длительностью в 600 секунд.

Таким образом, в эксперименте рассматриваются все пять типов шифрования (MD5, MD5 Base64, MD5 Unix, MD5 HMAC, MD5 AES), работающих с паролями различных длин с использованием букв, цифр и специальных символов. При этом каждый сервер (тип защиты) подвергается всем трем типам криптографического анализа (Brute-Force, Dictionary attack, Rainbow attack) одной длительности (10 минут).

3.4 Анализ результатов эксперимента по уязвимости

Наиболее уязвимым явилось хеширование без вторичного шифрования (75% потерянной информации). MD5 Base64, MD5 HMAC MD5 Unix потеряли 35,4%, 17,5% и 16,6% информационных битов соответственно. На рисунках 3.1, 3.2 показаны соотношения потерянных битов для вышеперечисленных алгоритмов. Результат доказывает гипотезу о недостаточной криптостойкости применяемого на сегодня алгоритма хеширования MD5.

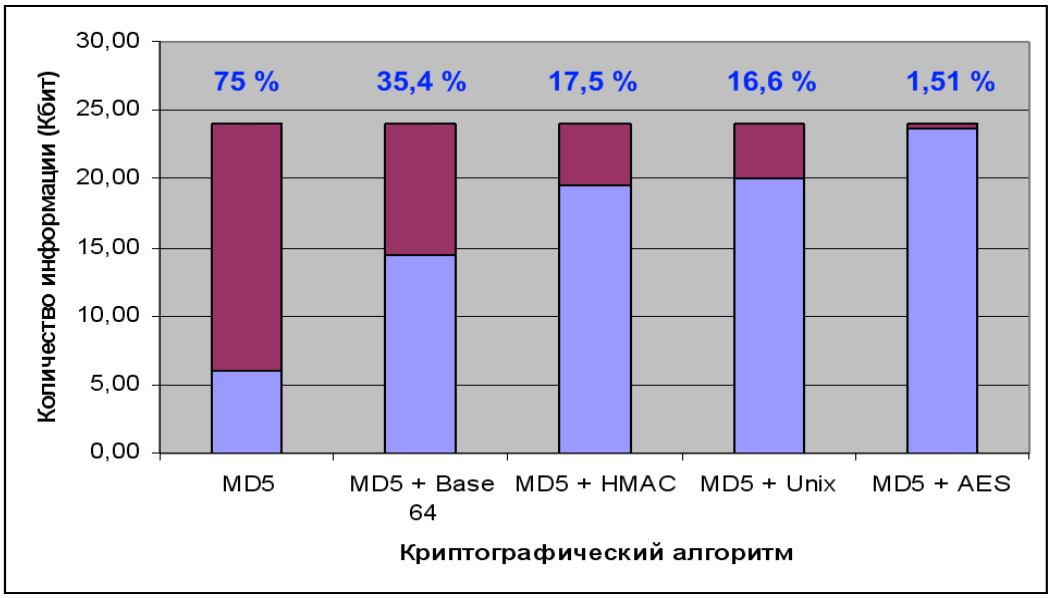


Рисунок 3.2 – Доля потерянной (красный маркер) в общем объеме переданной (синий маркер) информации

3.5 Анализ результатов эксперимента по эффективности от криптоаналитических атак

Вторым результатом стало выявление характеристик эффективности рассмотренных способов криптографического анализа. Были построены графики эффективности* криптоаналитических атак в зависимости от способа вторичного шифрования (рисунок 3.3), длины пароля и используемых в нем символов (рисунок 3.4). При этом из рисунка 3.3 можно говорить о линейном спаде эффективности все трех способ криптоанализа по мере движения от MD5 к MD5 AES. График подтверждает гипотезу о возможности увеличения криптостойкости путем вторичного шифрования хеш-сумм.

* За главный показатель эффективности метода бралась вероятность расшифровки бита информации P(x)

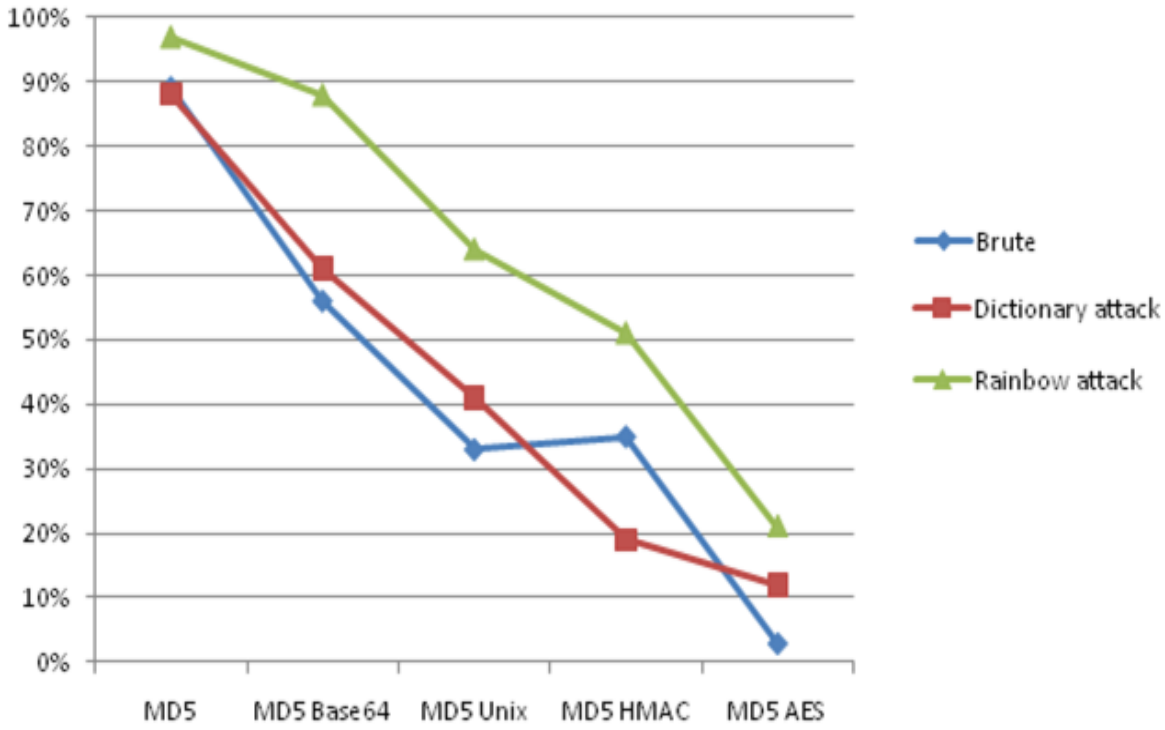


Рисунок 3.3 – Эффективность различных видов атак от способа шифрования

3.6 Анализ результатов эксперимента по эффективности различных видов атак от типа пароля

Из рисунка 3.4 можно заключить, что Brute attack наиболее эффективен при криптоанализе простых паролей (малой длины и без использования специальных символов). Dictionary attack следует использовать при расшифровке с ограничением по времени, когда в пароле присутствуют специальные символы. Наибольшей эффективностью обладает Rainbow attack. Данный метод было бы разумно применять при расшифровке сложных паролей с высокой криптостойкостью.

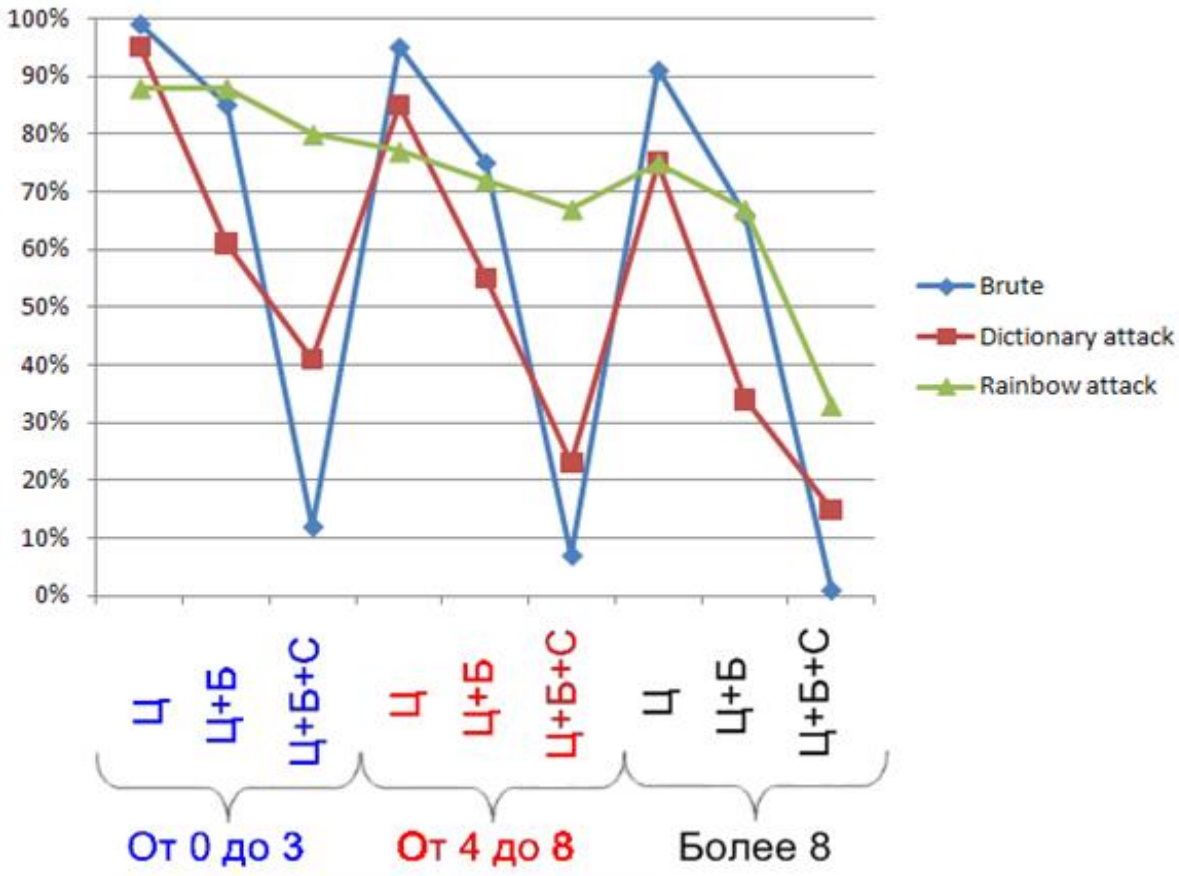


Рисунок 3.4 – Эффективность различных видов атак от типа пароля

3.7Анализ результатов эксперимента эффективности по времени

Следующим результатом эксперимента стало вычисление необходимы временных затрат, связанных с дополнительным шифрованием конечных сумм данных. На рисунке 3.5 показана гистограмма, иллюстрирующая временные задержки[**].

Из рисунка видно, что по мере увеличения криптостойкости алгоритмов, растет и время обработки информации. При этом алгоритм MD5 Base64 требует в 1,5 раза больше времени на обработку данных, алгоритм MD5 Unix – 2,25, MD5 HMAC – в 2,3 и MD5 AES – в 2,5 раза.

В таблице 3.1 приведены результаты проведенного эксперимента.

[**] На графике время показано в процентном соотношении, поскольку количественные показатели зависят от производительности конечного оборудования. При этом 100% времени бралось за временную задержку при шифровании MD5 без применения дополнительных алгоритмов.

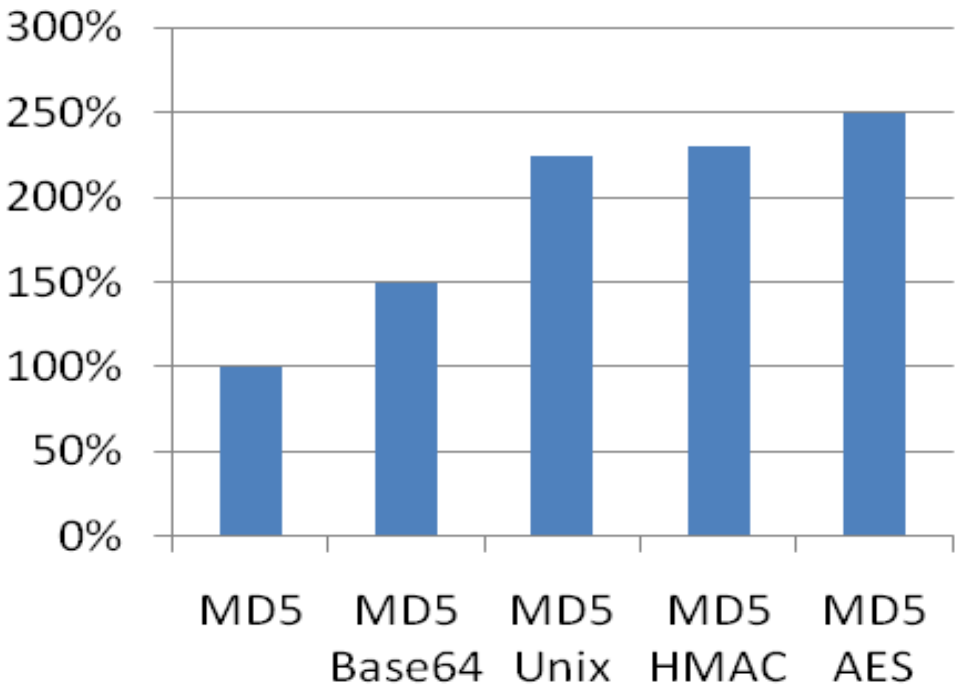


Рисунок 3.5 – Гистограмма дополнительных затрат времени на вторичное шифрование

3.8 Анализ результатов эксперимента надежности

По результатам эксперимента наиболее надежным алгоритмом шифрования стал MD5 AES (1,51% потерянных данных) рисунок 3.1.

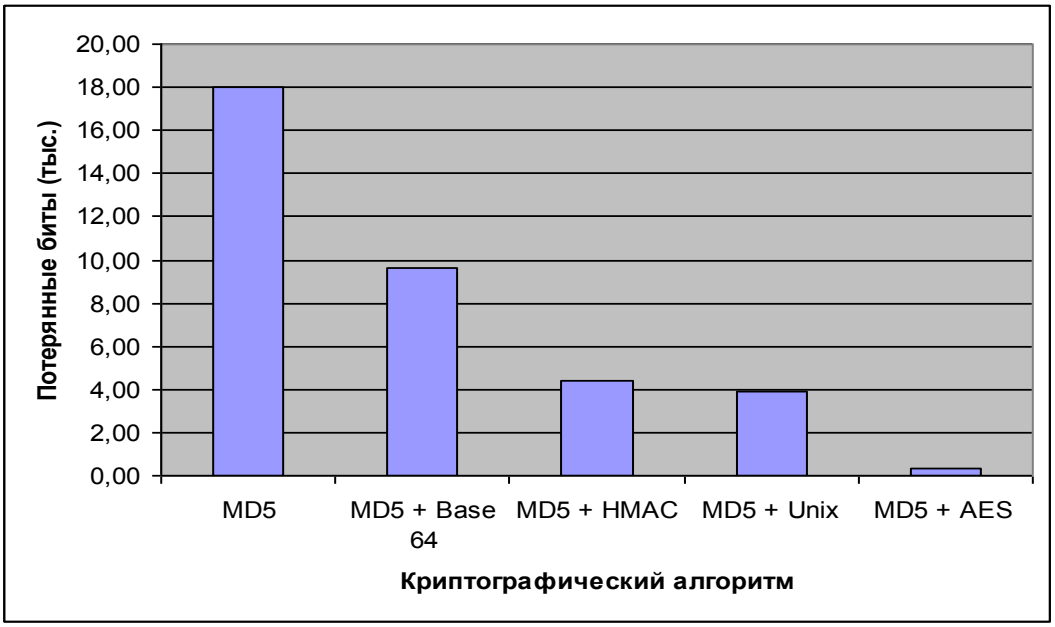


Рисунок 3.1 – Количество потерянной информации

Т а б л и ц а 3 . 1 – Результаты эксперимента

№ сервера	Тип применяемого шифрования	Тип пароля	Объем потерянной информации, Кбит	Объем сохраненной информации, Кбит	Эффективность Brute-Force атаки, %	Эффективность Dictionary атаки, %	Эффективность Rainbow атаки, %	Временная задержка, %
1	MD5	9 символов, только буквы	18,0	6,0	89	88	97	100
2	MD5 Base64	8 символов, цифры и буквы	8,496	15,504	56	61	88	15-
3	MD5 Unix	11 символов, только цифры	3,984	20,016	33	41	64	225
4	MD5 HMAC	4 символа, цифры, буквы и спец. символы	4,2	19,8	35	19	51	230
5	MD5 AES	12 символов, цифры, буквы и спец. символы	0,362	23,638	3	12	21	250

Приложение А

```
/*[abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  0 7  1][DABC  1 12  2][CDAB  2 17  3][BCDA  3 22  4]
[ABCD  4 7  5][DABC  5 12  6][CDAB  6 17  7][BCDA  7 22  8]
[ABCD  8 7  9][DABC  9 12 10][CDAB 10 17 11][BCDA 11 22 12]
[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]

/*[abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  1 5 17][DABC  6 9 18][CDAB 11 14 19][BCDA  0 20 20]
[ABCD  5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA  4 20 24]
[ABCD  9 5 25][DABC 14 9 26][CDAB  3 14 27][BCDA  8 20 28]
[ABCD 13 5 29][DABC  2 9 30][CDAB  7 14 31][BCDA 12 20 32]

/*[abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  5 4 33][DABC  8 11 34][CDAB 11 16 35][BCDA 14 23 36]
[ABCD  1 4 37][DABC  4 11 38][CDAB  7 16 39][BCDA 10 23 40]
[ABCD 13 4 41][DABC  0 11 42][CDAB  3 16 43][BCDA  6 23 44]
[ABCD  9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA  2 23 48]

/*[abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  0 6 49][DABC  7 10 50][CDAB 14 15 51][BCDA  5 21 52]
[ABCD 12 6 53][DABC  3 10 54][CDAB 10 15 55][BCDA  1 21 56]
[ABCD  8 6 57][DABC 15 10 58][CDAB  6 15 59][BCDA 13 21 60]
[ABCD  4 6 61][DABC 11 10 62][CDAB  2 15 63][BCDA  9 21 64]
```

Раунд 1

Раунд 2

Раунд 3

Раунд 4

Рисунок 1 – Четыре раунда преобразований в алгоритме MD5

Приложение Б

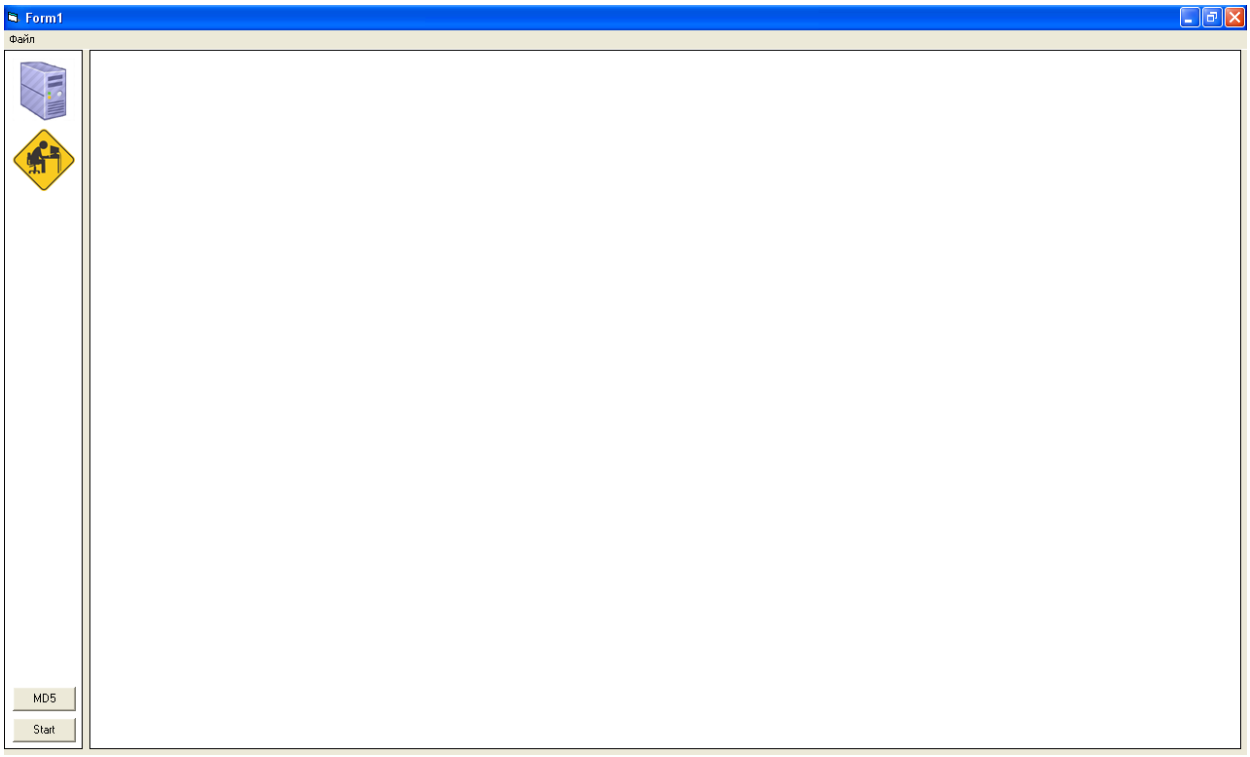


Рисунок 1 – Интерфейс программы

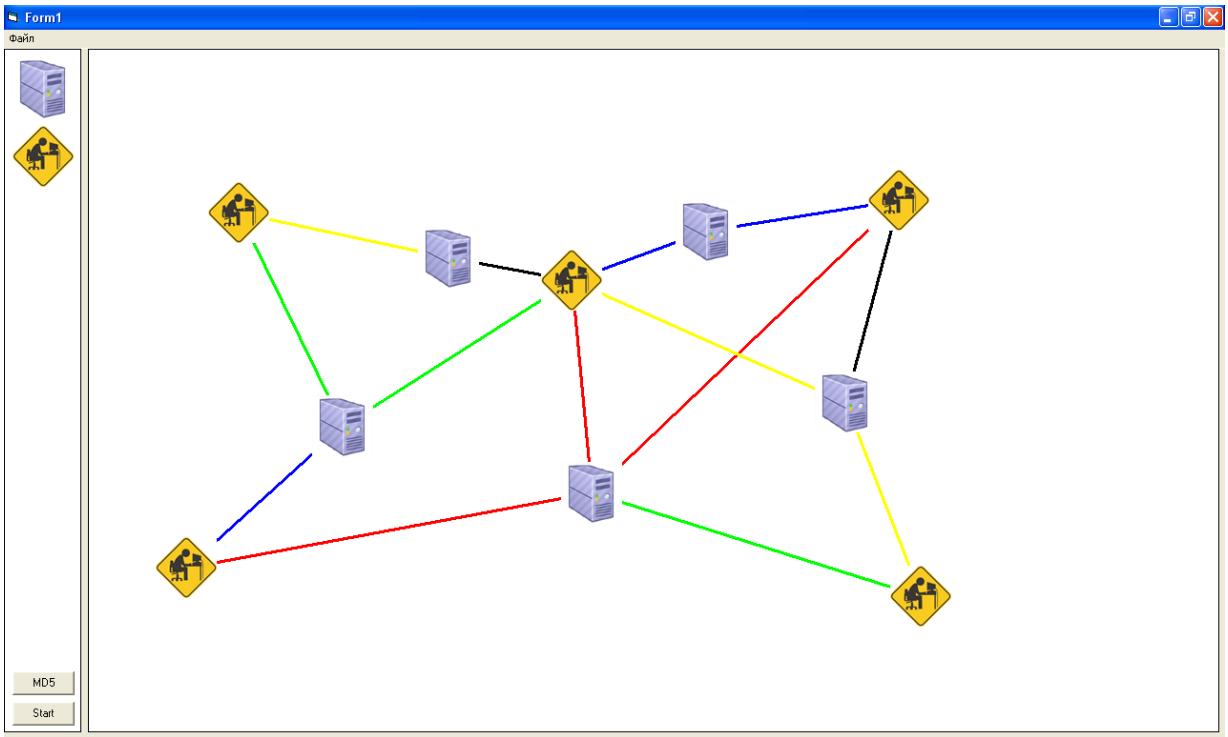


Рисунок 2 – Пример организации атак на серверы

Приложение В

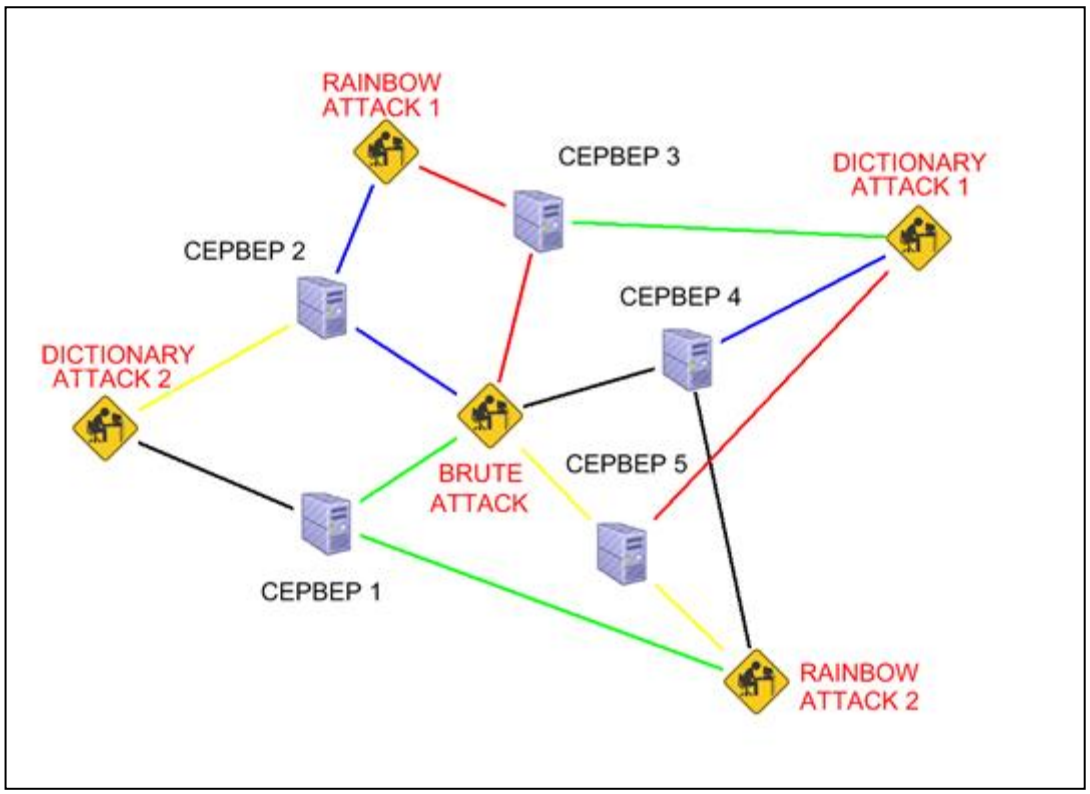


Рисунок 1 – Экспериментальная модель

Сервер #4
Шифр - MD5 + HMAC
Пароль - Цифры, буквы и спец.символы

Тип атаки	Время атаки	Потерянная информация
Rainbow	600	240000
Dictonary	600	150000
Brute	600	30000
Total	1800	420000

Рисунок 2 – Пример вывода результатов моделирования атаки на сервер

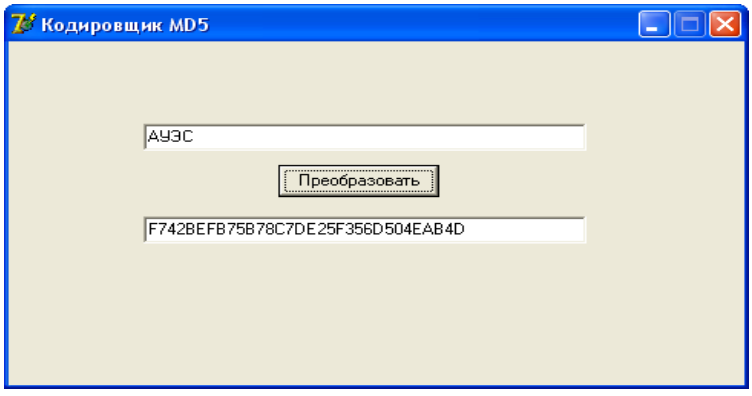


Рисунок 3 – Диалоговое окно разработанного Кодировщика MD5

Приложение Г

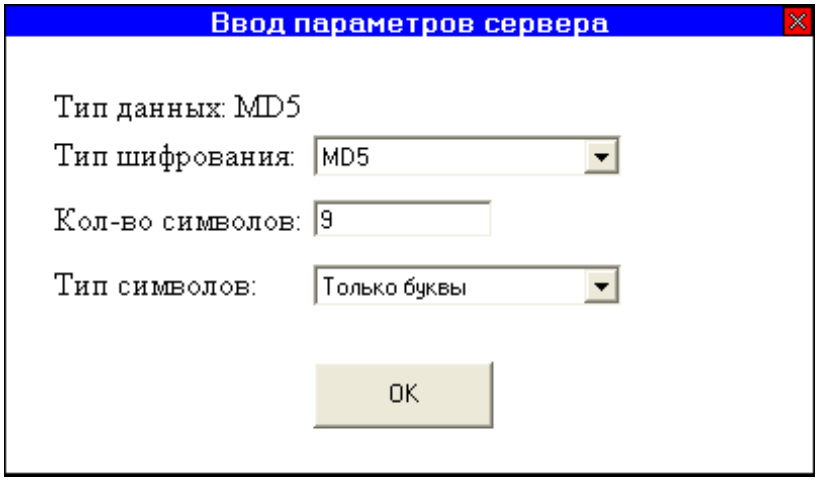


Рисунок 1 – Параметры сервера 1

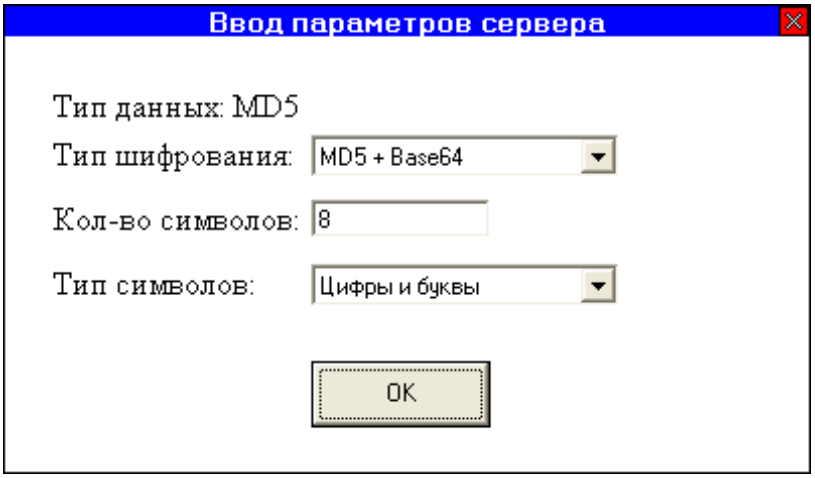


Рисунок 2 – Параметры сервера 2

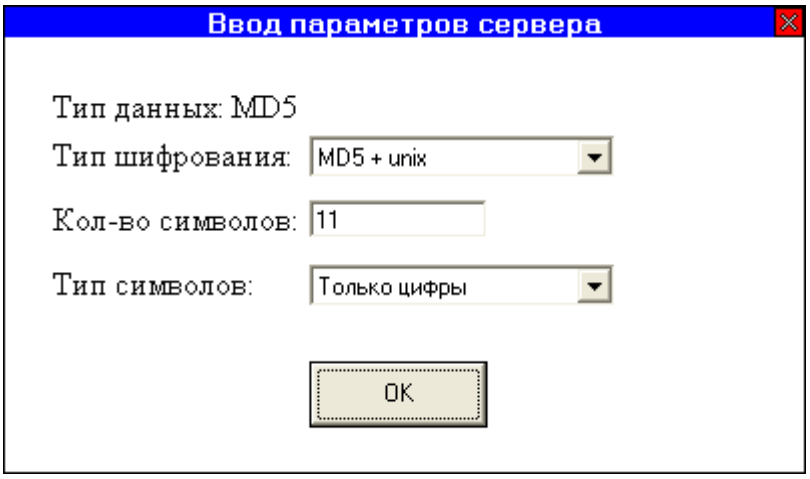


Рисунок 3 – Параметры сервера 3

Приложение Д

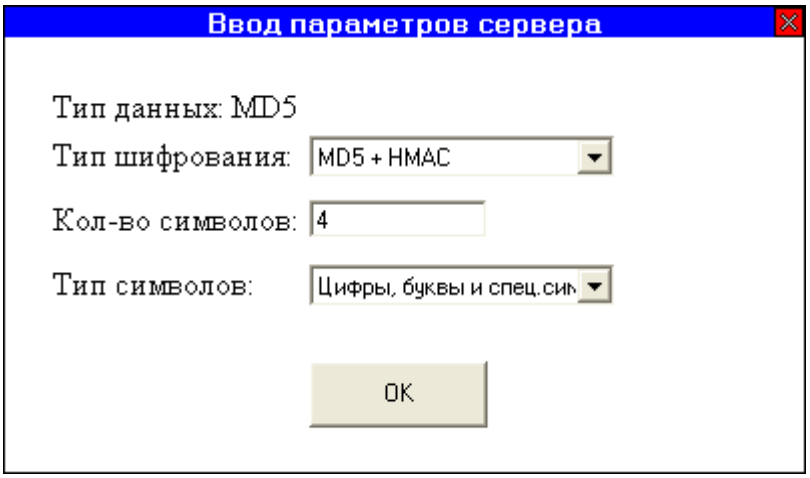


Рисунок 1 – Параметры сервера 4

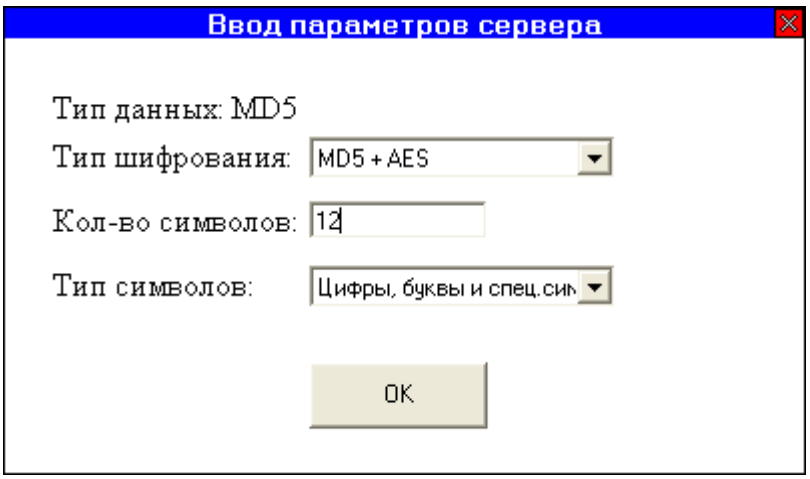


Рисунок 2 – Параметры сервера 5

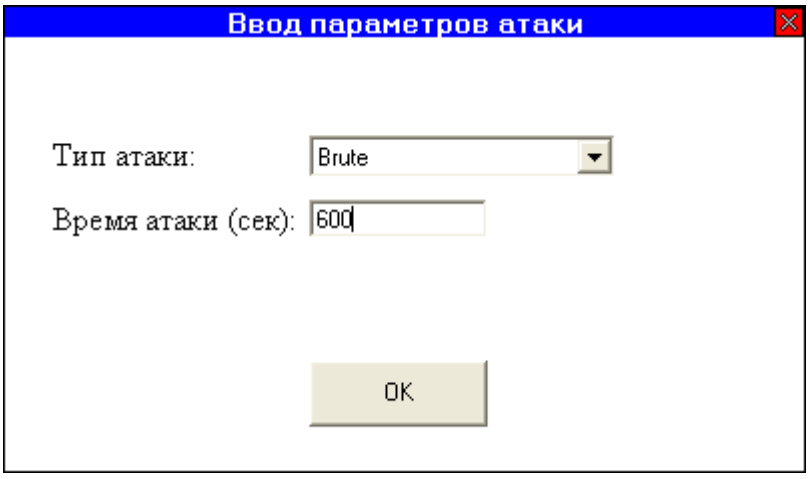


Рисунок 3 – Параметры атаки 1

Приложение Е

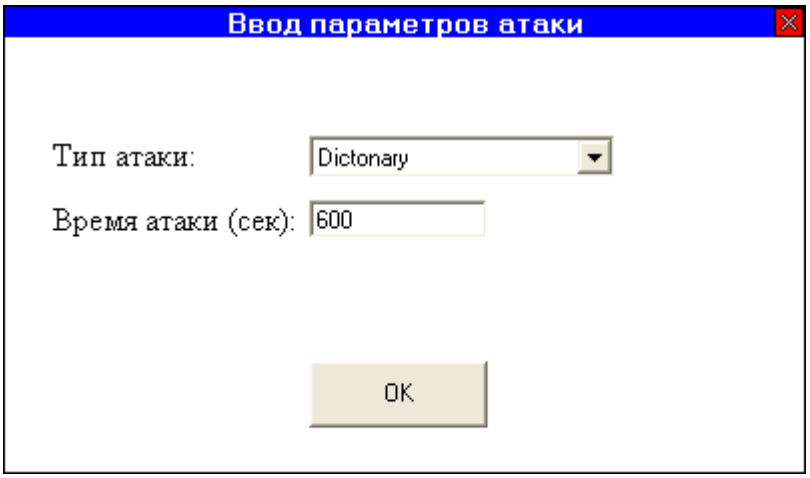


Рисунок 1 – Параметры атаки 2

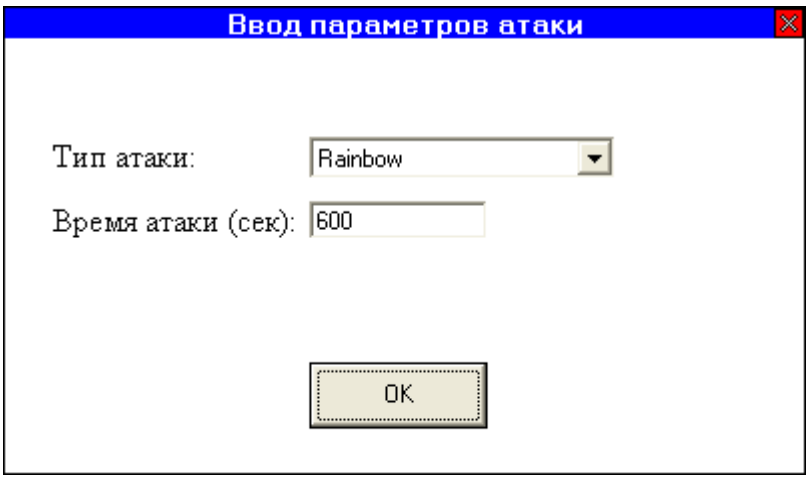


Рисунок 2 – Параметры атаки 3

Заключение

В данной работе были изучены характеристики алгоритмов AES, Serpent, Twofish, MARS и RC6.
Алгоритмы Serpent, Twofish, MARS и RC6 практически равнозначны по совокупности характеристик, за исключением алгоритма MARS, имеющего существенно больше недостатков, в том числе, алгоритм практически нереализуем в условиях ограниченных ресурсов.

Была изучена и доработана парольная система авторизации с применением технологии шифрования методом “нелинейной динамики” captcha. Так же алгоритм был модифицирован двухфакторной аутентификации с применением captcha.

В процессе исследования были изучены 5 основных видов криптографических алгоритмов, применяемых при хешировании MD5 в современных информационных сетях, оперирующих конфиденциальной информацией: MD5, MD5 Base 64, MD5 Unix, MD5 HMAC, MD5 AES.

Было разработано специальное программное обеспечение, которое позволяет создавать модели атак на серверы различными способами (Brute, Dictionary, Rainbow Attack), анализировать криптостойкость алгоритмов, используемых в современных информационных сетях, имеющих дело с данными аутентификации (пароль, имя учетной записи).

В эксперименте была рассмотрена модель атаки на серверы различными возможными способами. На каждом сервере были установлены уникальные параметры шифрования информации. Каждый сервер был подвергнут трем видам атак в течение 600 секунд. По итогам эксперимента самым эффективным оказался MD5 с шифрованием по алгоритму AES. Самый эффективный способ атаки – Rainbow Attack. Наименее криптостойкая система защиты – MD5 без дополнительного шифрования. При этом различные способы криптоанализа имеют разную эффективность в зависимости от способа защиты информации. Так, например, наиболее эффективным при шифровании MD5 является Brute, Dictionary attack имеет хорошую эффективность по взлому паролей с применением специальных символов, а Rainbow attack лучше применять при расшифровке сообщений со сложным шифрованием.

Однако, несмотря на все плюсы сложных криптографических алгоритмов (в частности MD5 Unix и MD5 HMAC), потери времени, которые возникают при их использовании, могут создать коллапс в информационных системах. Это, пожалуй, является основным препятствием в использовании вторичного шифрования. Ведь такие крупные системы, как ВКонтакте или Одноклассники не могут себе позволить увеличения времени обработки данных в 2 или 2,5 раза. Поэтому столь сложные алгоритмы было бы разумнее использовать в системах, требующих большого уровня защищенности конфиденциальных данных (например, в системах

электронной оплаты). А для сетей с большим количеством пользователей (социальные сети, почтовые сервисы) приемлемо было бы использование алгоритма MD5 Base64, обеспечивающего повышение криптографической стойкости в 2 раза по сравнению с хешированием MD5 без дополнительного шифрования (35,4% по сравнению с 75% потерянной информации). Время обработки информации при этом возрастает в 1,5 раза – приемлемые потери при двойном повышении защищенности данных.

Модель, разработанная в процессе исследования, полностью работоспособная и может использоваться для проведения дальнейших исследований. В будущем планируется подключение модулей взаимодействия с другим ПО, расширение возможности атак и защиты серверов, расширение функциональности (добавление возможности моделирования частных ЛВС, корпоративных сетей и т. д.), увеличение стабильности работы программы.

Список используемой литературы

1. AES Round 1 Information. // http://csrc.nist.gov — January 26, 2001.
2. Anderson R., Biham E. Two Practical and Provably Secure Block Ciphers: BEAR and LION. //http://citeseer.ist.psu.edu — 1995.
3. Anderson R., Biham E., Knudsen L. Serpent: A Proposal for the Advanced Encryption Standard. // http://csrc.nist.gov.
4. Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES). // http://csrc.nist.gov — Department of Commerce — National Institute of Standards and Technology — Federal Register: September 12, 1997.
5. Biham E., Dunkelman O., Keller N. Differential-Linear Cryptanalysis of Serpent. //http://citeseer.ist.psu.edu — Technion, Haifa, Israel.
6. Burwick C., Coppersmith D., D'Avignon E., Gennaro R., Halevi S., Jutla C., Matyas S.M.Jr., O'Connor L., Peyravian M., Safford D., Zunic N. MARS — a candidate cipher for AES. //http://www.ibm.com — IBM Corporation — Revised, September, 22 1999.
7. Courtois N., Castagnos G., Goubin L. What do DES S-boxes Say to Each Other? //http://eprint.iacr.org — Axalto Cryptographic Research & Advanced Security, Cedex, France.
8. Daemen J., Knudsen L., Rijmen V. The Block Cipher Square. //http://www.esat.kuleuven.ac.be.
9. FIPS 46-3. Data Encryption Standard (DES). // http://csrc.nist.gov — Reaffirmed 1999 October 25.
10. FIPS 81. DES Modes of Operation. // http://csrc.nist.gov — 1980 December 2.
11. Kocher P.C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. // http://citeseer.ist.psu.edu — Cryptography Research, Inc., San Francisco, USA.
12. Nechvatal J., Barker E., Bassham L., Burr W., Dworkin M., Foti J., Roback E. Report on the Development of the Advanced Encryption Standard (AES). // http://csrc.nist.gov — National Institute of Standards and Technology.
13. Nechvatal J., Barker E., Dodson D., Dworkin M., Foti J., Roback E. Status report on the first round of the development of the advanced encryption standard. // http://csrc.nist.gov — National Institute of Standards and Technology.
14. Rivest R.L., Robshaw M.J.B., Sidney R., Lin Y.L. The RC6 Block Cipher. //http://www.rsasecurity.com — Version 1.1 — August 20, 1998.
15. Schneier B. Description of a new variable-length key 64-bit block cipher (blowfish). //http://www.schneier.com.
16. Schneier B., Kelsey J., Whiting D., Wagner D., Hall C., Ferguson N. Twofish: A 128-bit Block Cipher. // http://www.schneier.com — 15 June 1998.
17. What are RC5 and RC6? // http://www.rsasecurity.com.

18. Панасенко С. Алгоритм шифрования DES и его варианты. // Connect! Мир связи. — 2006 — №№ 3-6.
19. Панасенко С. Интересные алгоритмы шифрования, часть 2. // BYTE/Россия. — 2006 — № 5 — с. 74-79.
20. Панасенко С.П., Батура В.П. Основы криптографии для экономистов: учебное пособие. Под ред. Л.Г. Гагариной. — М.: Финансы и статистика, 2005 — 176 с.
21. Соколов А.В., Шаньгин В.Ф. Защита информации в распределенных корпоративных сетях и системах. — М.: ДМК Пресс, 2002 — 656 с.
22. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. — Пер. с англ.: М.: Издательство ТРИУМФ, 2002 — 816 с.
23. Галатенко В.А., Основы информационной безопасности. – М.: ИНТУИТ.РУ «Интернет-Университет Информационных Технологий», 2003. – 280 стр.
24. Лапонина. О.Р. Криптографические основы безопасности. – М.: ИНТУИТ.РУ «Интернет- Университет Информационных Технологий», 2003. – 250 стр.