Алгоритм построения цифрового дайджеста MD4

Данная статья предлагает детальный анализ алгоритма построения цифрового дайджеста сообщения MD4. Теория, рассматриваемая в деталях, подкреплена весьма качественными примерами, реализованными на языках Си и Форт.

1. Введение

Язык программирования Си наиболее распространенный из языков на сегодняшний день. Но, как оказалось, в процессе написания статьи мне необходимо было выполнять некие вычисления. Язык Форт оказался здесь незаменимым помощником, он дал возможность создавать код, глубже вникать в алгоритм во время его реализации и выполнять этот алгоритм по шагам в интерактивном режиме. Лучше инструмента найти я не смог. Я использовал WinForth32, который можно бесплатно загрузить с адреса FTP://FTP.Taygeta.COM/pub/Forth/Compilers/Native/windows/Win32For/W32for42.exe Для тех, кто хочет познакомиться с Фортом поближе, я рекомендую посетить сайты HTTP://www.Forth.ORG и HTTP://www.Forth.ORG.RU.

Алгоритм MD4 это один из алгоритмов целого семейства, в этом семействе существует еще два алгоритма – MD2 и MD5. Но MD2 довольно старый алгоритм и сейчас практически не используется, а MD5 это следующий шаг после MD4, и в его основе лежит все тот же MD4. Поэтому я принял решение начать с более простого алгоритма, чтобы построить фундамент для понимания современных алгоритмов построения цифровых дайджестов.

2. Цифровой дайджест

Я прелагаю начать знакомство с цифровыми дайджестами с его определения, взятого из компьютерного словаря. Одно из определений звучит следующим образом: "Цифровой дайджест – это представление текста в виде строки цифр, построенной с использованием формулы, называемой односторонней хэш функцией. Шифрование цифрового дайджеста с помощью закрытого ключа создает цифровую подпись, которая представляет собой электронный вид аутентификации." И, наверное, будет неплохо привести пример. Вот цифровой дайджест строки "MD4" и файла WinWord.EXE построенный с помощью программы, взятой из документа RFC 1320:

MD4(MD4) = 9af4a3d5f2dfb4b6851b265b46fbeff3
MD4(WinWord.EXE) = 595452c1c9c3008273d83ad32cc8de16

Прошу заметить, что цифровой дайджест для файла WinWord.EXE будет отличаться для различных версий этого файла. Я строил дайджест для WinWord.EXE с характеристиками, представленными на Рисунке 1.

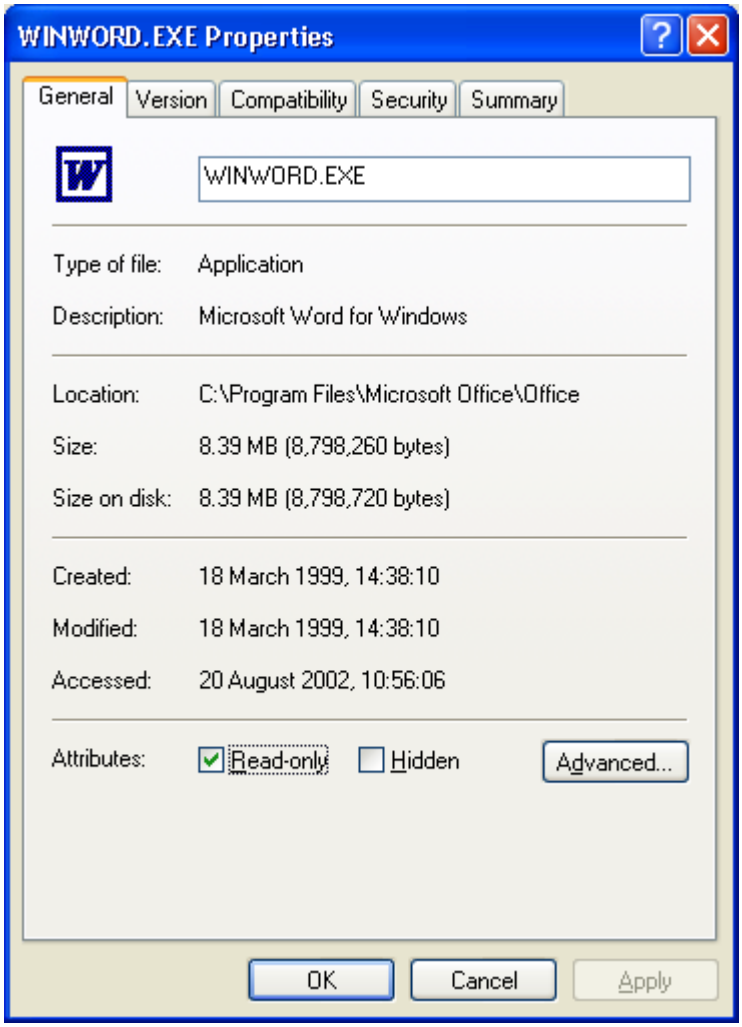

Рисунок 1. Диалоговое окно со свойствами фала WinWord.EXE.

Для начала освежим в памяти, что такое хэш. Хэш - это численная величина, полученная из текстовой строки. Эта величина существенно меньше, чем сама текстовая строка. Хэш строится с помощью одностороннего преобразования, или односторонней функции. Особенность этого преобразования заключается в том, что вероятность генерации одинакового хеш-значения для двух различных строк крайне низка. Процесс получения такой числовой величины из текстовой строки называется хэшированием. Хэширование играет большую роль в системах безопасности, где оно используется для того, чтобы удостовериться, что переданное сообщение не было подделано. Отправитель строит хэш сообщение, шифрует его и отправляет вместе с сообщением (также зашифрованным). Получатель дешифрует сообщение и хэш, строит новый хэш для дешифрованного сообщения и сравнивает его с полученным. Если они одинаковы, то вероятность того, что сообщение пришло без изменений очень высока. "Односторонняя" обозначает, что практически невозможно воспроизвести исходную текстовую строку на основании ее хэш значения. "Практически невозможно" в свою очередь обозначает, что для нахождения такой строки потребуется вычислительный ресурс недостижимой на данный момент мощности из-за ограниченности современных технологий, а также то, что время нахождения такой строки с использованием современных вычислительных ресурсов настолько велико (может измеряться годами), что даже если такая строка будет найдена, то ее информационный смысл уже утратит свою значимость.

3. Немного истории

Немного истории, конечно же, не помешает. Алгоритм MD4 был спроектирован профессором Ривестом из Массачусетского Технологического Института Соединенных Штатов Америки в 1990 году и является последователем алгоритма MD2. Этот алгоритм оптимизирован для быстрого построения цифрового дайджеста сообщения на 32-разрядных машинах. Описание алгоритма и пример реализации может быть найден в документе RFC 1320, перевод которого приведен в Приложении к этой статье. На сегодняшний день алгоритм MD4 считается взломанным.
В 1994 году Ван Ооршот и Вейнер оценили стоимость машины для поиска двух сообщений, приводящих к коллизии, то есть таких, которые генерируют одинаковый цифровой дайджест методом "грязной силы" для алгоритма MD5 – последователя алгоритма MD4. Эта оценка показала, что стоимость такой машины составляет 10 миллионов долларов (на 1994 год) и для нахождения коллизирующих сообщений в среднем требуется 24 дня работы.

4. MD4 процессор

В основе работы алгоритма MD4 лежит преобразование блока данных с использованием определенных формул. Но для начала посмотрим на внутреннюю структуру алгоритма как на специализированный процессор, который имеет свои регистры и операции. Начнем с регистров. На Рисунке 2 представлена внутренняя структура такого процессора.

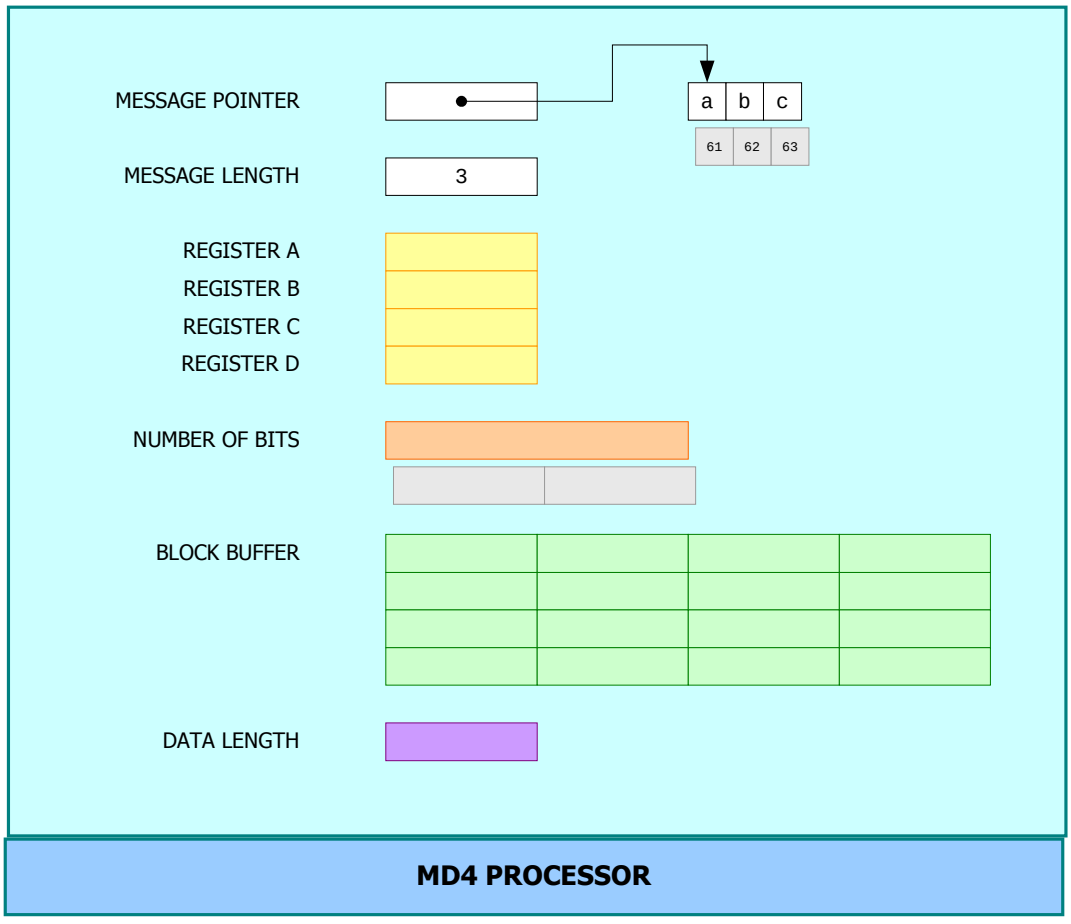

Рисунок 2. Структура MD4 процессора.

Два регистра "УКАЗАТЕЛЬ ДАННЫХ" и "РАЗМЕР ДАННЫХ" содержат, соответственно, указатель на буфер, где находится строка, подлежащая обработке и ее размер в байтах. На рисунке приведена строка, состоящая из трех символов, поэтому регистр "РАЗМЕР ДАННЫХ" содержит величину 3.
Регистры A, B, C и D содержат текущий дайджест обрабатываемого блока, а в конце обработки – цифровой дайджест сообщения. Каждый регистр содержит 32-разрядную беззнаковую величину, поэтому размер цифрового дайджеста будет равен 32*4=128 бит.
Регистр "РАЗМЕР СООБЩЕНИЯ" хранит размер обрабатываемого сообщения в битах. Эта величина понадобиться нам на последнем этапе работы алгоритма. Регистр хранит 64-разрядную величину.
Регистровая группа "БЛОЧНЫЙ БУФЕР" предназначена для хранения очередного блока, подлежащего обработке. Эта группа состоит из 16 регистров с номерами от 0 до 15.
Регистр "БАЙТ В БУФЕРЕ" хранит величину указывающую заполнение регистровой группы "БЛОЧНЫЙ БУФЕР".

4.1 Базовые операции MD4 процессора

Я начну с рассмотрения трех базовых функций, затем мы рассмотрим три базовых преобразования, которые выполняет процессор. И в конце мы объединим все это в операцию преобразования блока данных.
MD4 процессор реализует три базовых логических функции:

F(x,y,x) = (x AND y) OR (NOT x AND z)[1]
G(x,y,z) = (x AND y) OR (x AND z) OR (y AND z)
H(x,y,z) = x XOR y XOR z

Таблицы истинности этих логических функций выглядят следующим образом:

x	y	z	F
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**0**
1	1	0	**1**
1	1	1	**1**

x	y	z	G
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

x	y	z	H
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**1**

Каждая из функций получает на вход три 32-разрядные величины и формирует один 32-разрядный результат.

Здесь подошло время обратиться к исходному коду. В качестве иллюстрации мы будем использовать исходный код библиотеки OpenSSL (HTTP://www.OpenSSL.ORG). Вот как реализованы эти функции в этой библиотеке:

```
#define  F(b,c,d)  (((b) & (c)) | ((~(b)) & (d)))
#define  G(b,c,d)  (((b) & (c)) | ((b) & (d)) | ((c) & (d)))
#define  H(b,c,d)  ((b) ^ (c) ^ (d))
```

С целью изучения работы этих функций ниже приведен код на языке Форт. Его можно использовать для того, чтобы посмотреть какой результат получится после применения

[1] NOT x AND z = (NOT x) AND z

этих функций к каким-либо данным. (Примечание: если хотите увидеть результат в двоичном виде, то замените слово HEX на слово BINARY.)

```
: D-FF-BC ( c b -- x )
   ?PRINT IF 2DUP CR ." X & Y = " 8 HEX U.R SPACE ." & " 8 HEX U.R THEN
   AND
   ?PRINT IF DUP SPACE ." = " 8 HEX U.R THEN
   ;
: D-FF-~B ( b -- ~b )
   ?PRINT IF DUP CR ." ~X = ~" 8 HEX U.R THEN
   INVERT
   ?PRINT IF DUP SPACE ." = " 8 HEX U.R THEN
   ;
: D-FF-~BD ( ~b d -- x )
   ?PRINT IF 2DUP SWAP CR ." ~X & Z = " 8 HEX U.R SPACE ." & " 8 HEX U.R THEN
   AND
   ?PRINT IF DUP SPACE ." = " 8 HEX U.R THEN
   ;
: D-FF-F ( x1 x2 -- x3 )
   ?PRINT IF 2DUP SWAP CR ." . | .. = " 8 HEX U.R SPACE ." | " 8 HEX U.R THEN
   OR
   ?PRINT IF DUP SPACE ." = " 8 HEX U.R THEN
   ;
: D-FF ( b-addr c-addr d-addr -- f )
   @ ROT @ ROT @ ( d b c )
   OVER    ( d b c b )
   D-FF-BC  ( d b xi )
   SWAP    ( d xi b )
   D-FF-~B  ( d xi ~b )
   ROT     ( xi ~b d )
   D-FF-~BD ( xi xii )
   D-FF-F   ( f )
   ;

: D-FG-CD ( c d -- x )
   ?PRINT IF 2DUP CR ." Y & Z = " SWAP 8 U.R SPACE ." & " 8 U.R THEN
   AND
   ?PRINT IF DUP SPACE ." = " 8 U.R THEN
   ;
: D-FG-BD ( d b -- x )
   ?PRINT IF 2DUP CR ." X & Z = " SWAP 8 U.R SPACE ." & " 8 U.R THEN
   AND
   ?PRINT IF DUP SPACE ." = " 8 U.R THEN
   ;
: D-FG-CDvBD ( cd bd -- x )
   ?PRINT IF 2DUP CR ." YZ | XZ = " SWAP 8 U.R SPACE ." | " 8 U.R THEN
   OR
   ?PRINT IF DUP SPACE ." = " 8 U.R THEN
   ;
: D-FG-BC ( b c -- x )
   ?PRINT IF 2DUP CR ." X & Y = " SWAP 8 U.R SPACE ." & " 8 U.R THEN
   AND
   ?PRINT IF DUP SPACE ." = " 8 U.R THEN
   ;
: D-FG-vBC ( x1 bc -- x2 )
   ?PRINT IF 2DUP CR ." . | XY = " SWAP 8 U.R SPACE ." | " 8 U.R THEN
   OR
   ?PRINT IF DUP SPACE ." = " 8 U.R THEN
   ;
: D-FG ( b-addr c-addr d-addr -- g )
   @ ROT @ ROT @ ROT
   3DUP      ( b c d b c d )
   D-FG-CD   ( b c d b xi )
   -ROT      ( b c xi d b )
   D-FG-BD   ( b c xi xii )
```

```
   D-FG-CDvBD ( b c xiii )
   -ROT       ( xiii b c )
   D-FG-BC    ( xiii xiv )
   D-FG-vBC   ( g )
   ;
```

Приведенный код может показаться достаточно длинным и запутанным, но он производит вывод большого количества информации. Кроме того, его можно существенно упростить (оставим это в качестве упражнения для читателя). А также хочу обратить внимание читателя на то, что все слова в этом коде начинаются с префикса "D-", то есть "DEBUG" – ОТЛАДОЧНЫЙ. Вот для сравнения код, из которого были удалены все операторы отладочной печати:

```
: FF ( b-addr c-addr d-addr -- f ) @ ROT @ ROT @ OVER AND SWAP INVERT ROT AND OR ;
: FG ( b-addr c-addr d-addr -- g ) @ ROT @ ROT @ ROT 3DUP AND -ROT AND OR -ROT AND OR ;
: FH ( b-addr c-addr d-addr -- h ) @ ROT @ ROT @ XOR XOR ;
```

Не правда ли, гораздо проще!

Процессор также выполняет три базовые операции преобразования. Эти операции используют определенные выше функции F, G и H и выражаются следующими формулами:

a = (a + F(b,c,d) + X[k]) <<< s
a = (a + G(b,c,d) + X[k] + 0x5A827999) <<< s
a = (a + H(b,c,d) + X[k] + 0x6ED9EBA1) <<< s

где, a, b, c, d обозначают регистры A, B, C, D; F, G и H рассмотренные выше функции; <<< операцию циклического сдвига влево; s – количество разрядов сдвига.

Как и в предыдущем случае рассмотрим как эти операции реализованы на языке программирования Си в библиотеке OpenSSL:

```
#define R0(a,b,c,d,k,s,t) { \
        a+=((k)+(t)+F((b),(c),(d))); \
        a=ROTATE(a,s); };

#define R1(a,b,c,d,k,s,t) { \
        a+=((k)+(t)+G((b),(c),(d))); \
        a=ROTATE(a,s); };\

#define R2(a,b,c,d,k,s,t) { \
        a+=((k)+(t)+H((b),(c),(d))); \
        a=ROTATE(a,s); };
```

А вот и код на языке Форт для цели изучения работы алгоритма, к чему мы перейдем в ближайшее время:

```
: D-TRANSFORM-F+XI ( f nx -- x )
   ?PRINT IF 2DUP CR ." F + X[" 2 DECIMAL .R ." ] = " 8 HEX U.R THEN
   CELLS  ( f nxi )
   MD4_CTX_DATA
   +      ( f a-nxi )
   @      ( f xi )
   ?PRINT IF DUP SPACE ." + " 8 HEX U.R THEN
   +      ( x2 )
```

```
    ?PRINT IF DUP SPACE ." = " 8 HEX U.R THEN
    ;
: D-TRANSFORM-+N ( x1 x2 -- x3 )
    ?PRINT IF 2DUP CR ." . + N = " 8 HEX U.R SPACE ." + " 8 HEX U.R THEN
    +
    ?PRINT IF DUP SPACE ." = " 8 HEX U.R THEN
    ;
: D-TRANSFORM-+A ( x1 x2 -- x3 )
    ?PRINT IF 2DUP CR ." . + A = " 8 HEX U.R SPACE ." + " 8 HEX U.R THEN
    +
    ?PRINT IF DUP SPACE ." = " 8 HEX U.R THEN
    ;
: D-TRANSFORM-ROL ( ns x1 -- x2 )
    ?PRINT IF 2DUP CR ." . <<< " OVER 2 DECIMAL .R SPACE ." = " 8 HEX U.R SPACE ." <<< " 2
DECIMAL .R THEN
    SWAP
    LROTATE
    ?PRINT IF DUP SPACE ." = " 8 HEX U.R THEN
    ;
: D-TRANSFORM ( ns a-addr x f nx -- ai )
    \ Execute F + X[I]
    D-TRANSFORM-F+XI ( ns a-addr x xi )
    \ Execute . + N
    D-TRANSFORM-+N   ( ns a-addr xii )
    \ Execute . + A
    SWAP @ SWAP      ( ns a xii )
    D-TRANSFORM-+A   ( ns xiii )
    \ Execute . <<< S
    D-TRANSFORM-ROL  ( ai )
    ;
```

И ее аналог для рабочей версии:

```
: TRANSFORM ( ns a-addr x f nx -- ai ) CELLS MD4_CTX_DATA + @ + + SWAP @ + SWAP
LROTATE ;
```

Теперь, когда мы познакомились с базовыми функциями и базовыми операциями MD4 процессора, мы готовы перейти к изучению его работы. Однако, перед тем как начать рассмотрение этого процесса, необходимо сказать несколько слов об инициализации процессора.

4.2 Инициализация процессора.

Как и любой другой процессор, аппаратный или программный, наш MD4 процессор требует инициализации, то есть занесения начальных данных в регистры перед началом работы. Алгоритм построения цифрового дайджеста MD4 требует, чтобы регистры процессора были инициализированы таким образом, как это показано на Рисунке 3.

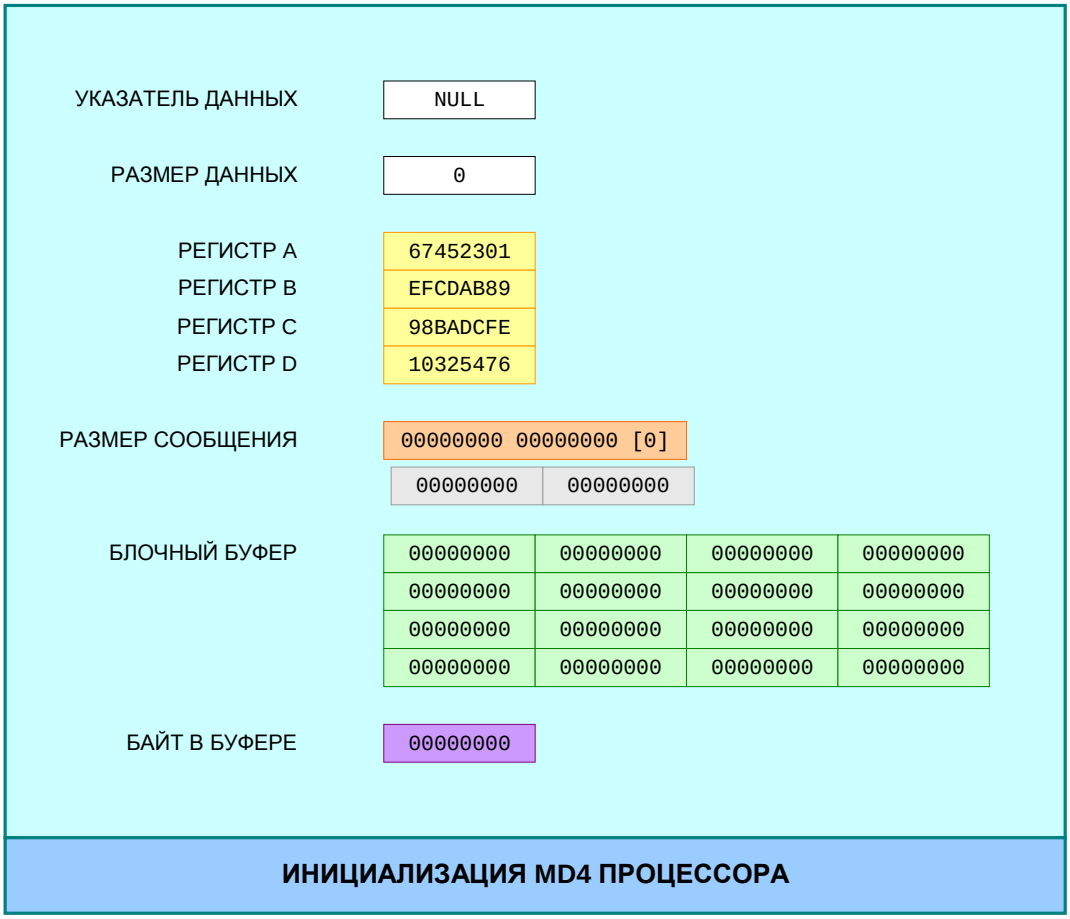

Рисунок 3. Состояние MD4 процессора после инициализации.

Откуда берут происхождение магические константы в регистрах A, B, C и D неизвестно. Скорее всего это знает изобретатель алгоритма Ривест или об этом сказано где-то в учебниках по криптографии.

Все остальные регистры должны быть инициализированы нулевыми значениями.

4.3 Работа MD4 процессора

Процессор выполняет четыре базовые операции, которые преобразуют содержимое регистров процессора: операция преобразование регистра A, операция преобразование регистра B, операция преобразование регистра C, и операция преобразование регистра D.

Операция

a = (a + F(b,c,d) + X[k]) <<< s

в применении к конкретным регистрам имеет следующий вид

A = (A + F(B,C,D) + X[0]) <<< 3

То есть процессор на основании содержимого регистров A, B, C и D, а также слова 0 из "БЛОЧНОГО БУФЕРА" строит новое значение регистра A (см. Рисунок 4)

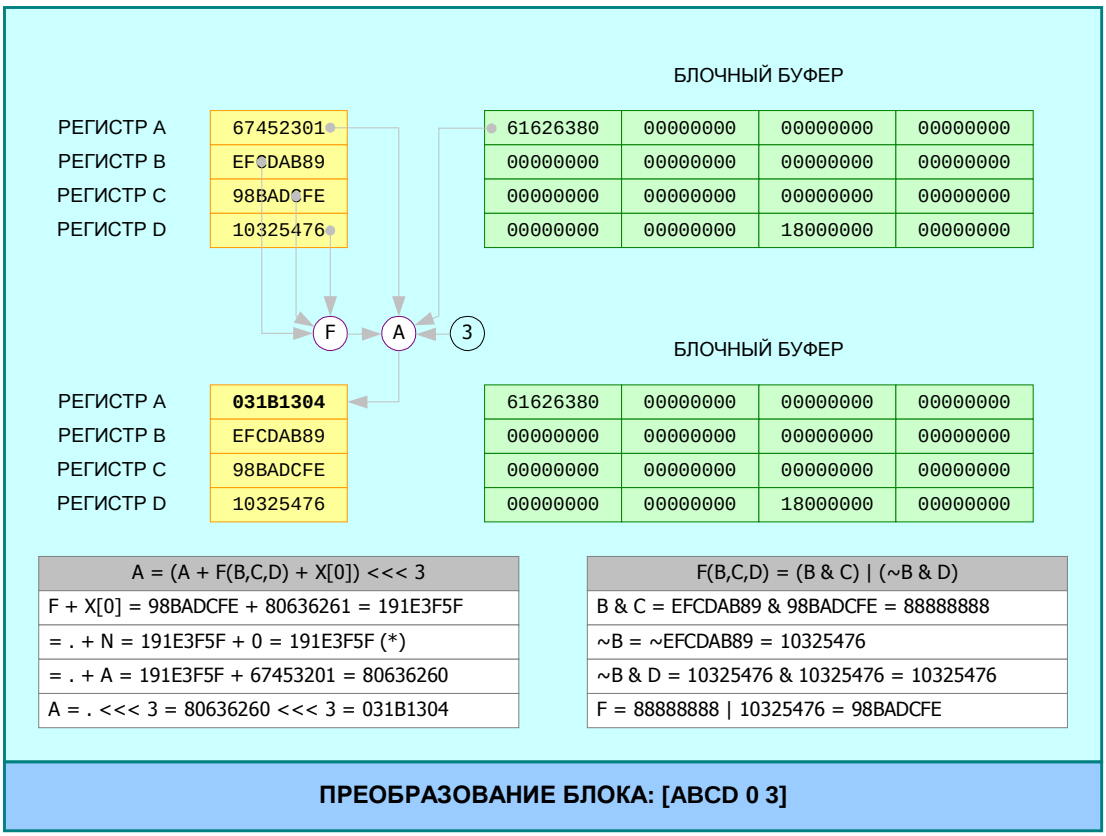

Рисунок 4. Базовая операция преобразования регистра A

Функция F вычисляется на основании значений из трех регистров процессора B, C и D. Пошаговое вычисление представлено на рисунке и состоит из четырех операций: B & C, затем ~B, затем ~B & D, и, наконец операция ИЛИ над двумя предыдущими промежуточными результатами. Тот же самый результат можно получить в результате применения логической операции предоставленной таблицей истинности для функции F, что позволяет реализовать ее в аппаратном виде.

Далее вычисленное значение функции F используется в формуле преобразования для получения нового значения регистра A. Эта операция также может быть разбита на элементарные операции: вначале выполняется сложение вычисленного значения функции F со значением элемента БЛОЧНЫЙ БУФЕР [0], затем к нему добавляется числовая константа (в данном случае это константа 0, поэтому операцию можно опустить), затем содержимое регистра A. Полученный результат сдвигается циклически влево на 3 разряда, давая в результате новое значение регистра A.

В применении к регистру D операция преобразования выглядит следующим образом (см. Рисунок 5):

D = (D + F(A,B,C) + X[1]) <<< 7

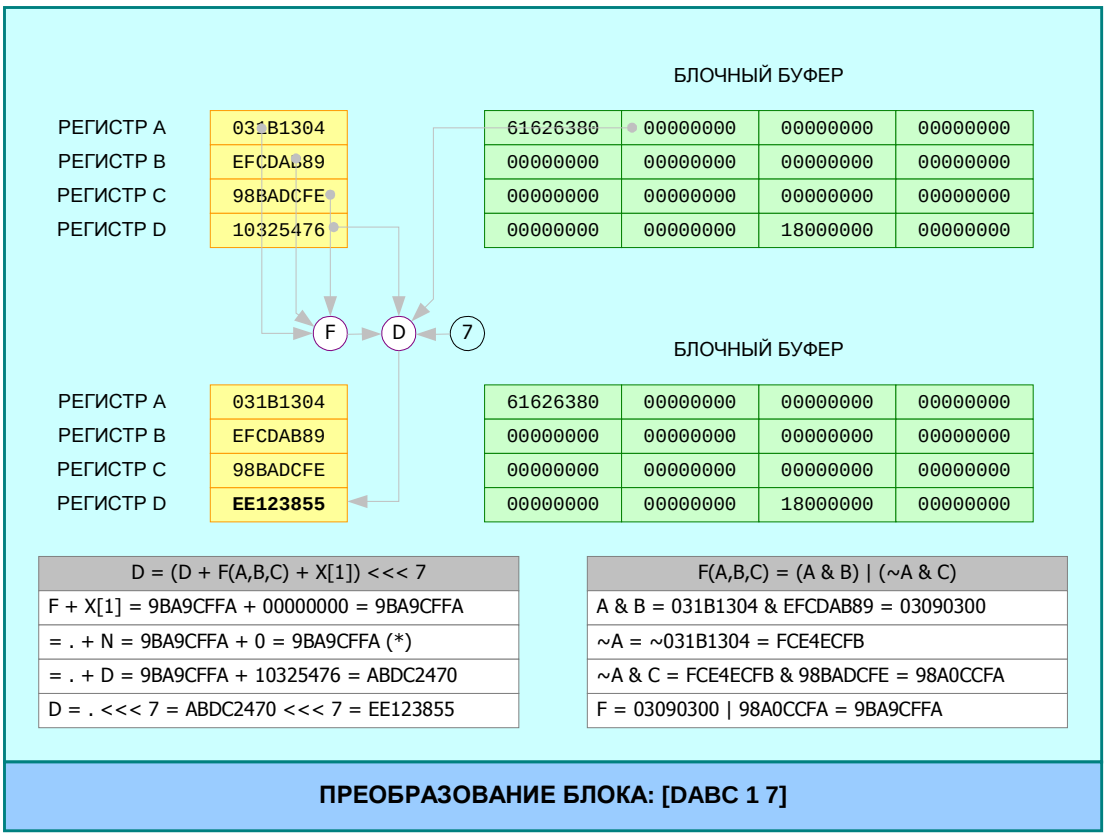

Рисунок 5. Базовая операция преобразования регистра D

Теперь мы подошли к вопросу о последовательности элементарных преобразований.

Последовательность элементарных преобразований называется Раундом. MD4 Процессор выполняет три раунда: Раунд 1, Раунд 2 и Раунд 3. Раунд 1 использует базовую функцию F. Раунд 2 использует базовую функцию G. Раунд 3 использует базовую функцию H. Все раунды описаны в документе RFC 1320.

Мы рассмотрим Раунд 1 и его реализацию. Раунд 1 представляет собой последовательность элементарных операций преобразования регистров процессора MD4 на основании предыдущего содержимого регистров и данных из БЛОЧНОГО БУФЕРА.

Пусть [abcd k s] обозначает базовую операцию a = (a + F(b,c,d) + X[k]) <<< s. Тогда Раунд 1 представляет собой следующую последовательность элементарных преобразований:

[ABCD 0 3] [DABC 1 7] [CDAB 2 11] [BCDA 3 19]
[ABCD 4 3] [DABC 5 7] [CDAB 6 11] [BCDA 7 19]
[ABCD 8 3] [DABC 9 7] [CDAB 10 11] [BCDA 11 19]
[ABCD 12 3] [DABC 13 7] [CDAB 14 11] [BCDA 15 19]

На языке программирования Си эта последовательность в библиотеке OpenSSL реализована следующим образом:

```
/* Round 0 */
R0(A,B,C,D,X[ 0], 3,0);
R0(D,A,B,C,X[ 1], 7,0);
R0(C,D,A,B,X[ 2],11,0);
R0(B,C,D,A,X[ 3],19,0);
R0(A,B,C,D,X[ 4], 3,0);
```

```
R0(D,A,B,C,X[ 5], 7,0);
R0(C,D,A,B,X[ 6],11,0);
R0(B,C,D,A,X[ 7],19,0);
R0(A,B,C,D,X[ 8], 3,0);
R0(D,A,B,C,X[ 9], 7,0);
R0(C,D,A,B,X[10],11,0);
R0(B,C,D,A,X[11],19,0);
R0(A,B,C,D,X[12], 3,0);
R0(D,A,B,C,X[13], 7,0);
R0(C,D,A,B,X[14],11,0);
R0(B,C,D,A,X[15],19,0);
```

Все очень просто и даже не требует никаких комментариев.

Для изучения работы я использовал простой код на языке Форт:

```
: D-STEPF1  ( -- )  3 RA 0 RB RC RD D-FF  0 D-TRANSFORM RA !  ?PRINT IF .ABCD CR CR THEN ;
: D-STEPF2  ( -- )  7 RD 0 RA RB RC D-FF  1 D-TRANSFORM RD !  ?PRINT IF .ABCD CR CR THEN ;
: D-STEPF3  ( -- ) 11 RC 0 RD RA RB D-FF  2 D-TRANSFORM RC !  ?PRINT IF .ABCD CR CR THEN ;
: D-STEPF4  ( -- ) 19 RB 0 RC RD RA D-FF  3 D-TRANSFORM RB !  ?PRINT IF .ABCD CR CR THEN ;

: D-STEPF5  ( -- )  3 RA 0 RB RC RD D-FF  4 D-TRANSFORM RA !  ?PRINT IF .ABCD CR CR THEN ;
: D-STEPF6  ( -- )  7 RD 0 RA RB RC D-FF  5 D-TRANSFORM RD !  ?PRINT IF .ABCD CR CR THEN ;
: D-STEPF7  ( -- ) 11 RC 0 RD RA RB D-FF  6 D-TRANSFORM RC !  ?PRINT IF .ABCD CR CR THEN ;
: D-STEPF8  ( -- ) 19 RB 0 RC RD RA D-FF  7 D-TRANSFORM RB !  ?PRINT IF .ABCD CR CR THEN ;

: D-STEPF9  ( -- )  3 RA 0 RB RC RD D-FF  8 D-TRANSFORM RA !  ?PRINT IF .ABCD CR CR THEN ;
: D-STEPF10 ( -- )  7 RD 0 RA RB RC D-FF  9 D-TRANSFORM RD !  ?PRINT IF .ABCD CR CR THEN ;
: D-STEPF11 ( -- ) 11 RC 0 RD RA RB D-FF 10 D-TRANSFORM RC !  ?PRINT IF .ABCD CR CR THEN
;
: D-STEPF12 ( -- ) 19 RB 0 RC RD RA D-FF 11 D-TRANSFORM RB !  ?PRINT IF .ABCD CR CR THEN
;

: D-STEPF13 ( -- )  3 RA 0 RB RC RD D-FF 12 D-TRANSFORM RA !  ?PRINT IF .ABCD CR CR THEN
;
: D-STEPF14 ( -- )  7 RD 0 RA RB RC D-FF 13 D-TRANSFORM RD !  ?PRINT IF .ABCD CR CR THEN
;
: D-STEPF15 ( -- ) 11 RC 0 RD RA RB D-FF 14 D-TRANSFORM RC !  ?PRINT IF .ABCD CR CR THEN
;
: D-STEPF16 ( -- ) 19 RB 0 RC RD RA D-FF 15 D-TRANSFORM RB !  ?PRINT IF .ABCD CR CR THEN
;

: D-ROUND1 ( -- )
   D-STEPF1  D-STEPF2  D-STEPF3  D-STEPF4
   D-STEPF5  D-STEPF6  D-STEPF7  D-STEPF8
   D-STEPF9  D-STEPF10 D-STEPF11 D-STEPF12
   D-STEPF13 D-STEPF14 D-STEPF15 D-STEPF16
   ;
```

Здесь больше кода, но не забывайте – он производит вывод промежуточной информации для анализа работы алгоритма. Рабочий код выглядит проще.

Три раунда "запутывают" сообщение и "сжимают" его, формируя таки образом цифровой дайджест. Интересно взглянуть на то в какой последовательности обрабатываются ячейки БЛОЧНОГО БУФЕРА на каждом раунде. Следующие три рисунка иллюстрируют этот процесс.

Рисунок 6. Последовательность обработки на Раунде 1.

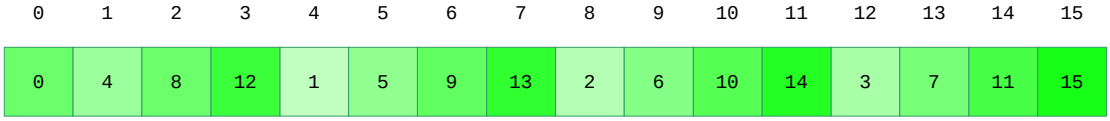

Рисунок 7. Последовательность обработки на Раунде 2.

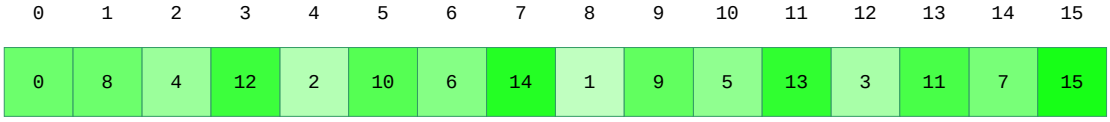

Рисунок 8. Последовательность обработки на Раунде 3.

5. Разбивка сообщения на блоки

Таким образом, сообщение разбивается на блоки по 64 символа, и каждый блок обрабатывается описанным выше способом, давая в результате цифровой дайджест сообщения. Возникает только один вопрос: что делать с последним блоком? Ведь он может и не содержать 64 символа (а чаще всего так и бывает).

Для того, чтобы разрешить эту ситуацию, все сообщение расширяется до некоего размера. Это размер сообщения в битах должен быть конгруэнтен 448 по модулю 512. Что это значит? Это значит, что если к размеру сообщения в битах после расширения добавить 64 (= 512 – 448), то полученная величина должна быть кратна 512, то есть делиться на 512 без остатка.

Рассмотрим пример. Пусть дано сообщение "abc". Его длина в битах равна 3 * 8 = 24. Расширив сообщение до 448 бит мы и получим выполнение требуемого условия: 448 + 64 = 512; 512 / 512 = 1 остаток 0.

6. Расширение сообщения.

Сообщение расширяется следующим образом. В конец сообщения добавляется единичный бит, а потом добавляется столько нулевых бит, чтобы длина полученного расширенного сообщения была конгруэнтна 448 по модулю 512.

Продолжая рассматривать пример сообщения "abc", расширим его, добавив единичный бит и 448 – 24 – 1 = 423 нулевых бита. В результате получим блок расширенного сообщения, приведенный на рисунках 4 и 5.

"Но для чего же предназначены последние 64 бита в блоке?" -- спросит внимательный читатель. Они предназначены для дальнейшего расширения сообщения, теперь уже до размера кратного размеру блока, то есть 512 бит. Теперь мы расширяем сообщение его длиной в битах ДО расширения.

В нашем случае мы должны расширить сообщение величиной 24 (3 * 8 = 24) представленной в 64-разрядном виде. Итак, 24 (10) = 0000 0000 0000 0018 (16). Вначале дописывается младшее 32-разрядное слово, а затем старшее.

Полученный таким образом блок обрабатывается рассмотренным ранее способом.

В этом процессе расширения сообщения есть один нетривиальный, на первый взгляд, случай – это когда длина исходного сообщения равна 56 символам (или, точнее, 56 байтам).

В этом случае необходимо добавить к битовому представлению исходного сообщения единичный бит, а затем 32*2-1 = 8*8-1 = 63 нулевых бита и преобразовать полученный таким образом заполненный БЛОЧНЫЙ БУФЕР. После этого нужно сформировать еще один БЛОЧНЫЙ БУФЕР, заполненный (16-2)*32 = 448 нулевыми битами и затем расширить его длиной исходного сообщения в битах ДО расширения. И снова преобразовать полученный блок рассмотренным ранее способом.

7. Построение цифрового дайджеста строки

Имея набор примитивов для обработки сообщения по блокам, мы можем построить программу для построения цифрового дайджеста строки символов.

Вначале глянем на то, как реализованы эти примитивы в библиотеке OpenSSL:

- инициализация процессора

```
#define INIT_DATA_A (unsigned long)0x67452301L
#define INIT_DATA_B (unsigned long)0xefcdab89L
#define INIT_DATA_C (unsigned long)0x98badcfeL
#define INIT_DATA_D (unsigned long)0x10325476L

void MD4_Init(MD4_CTX *c)
        {
        c->A=INIT_DATA_A;
        c->B=INIT_DATA_B;
        c->C=INIT_DATA_C;
        c->D=INIT_DATA_D;
        c->Nl=0;
        c->Nh=0;
        c->num=0;
        }
```

- обработка сообщения

```
void MD4_Update(MD4_CTX *c, const void *data_, unsigned long len)
        {
        const unsigned char *data=data_;
        register HASH_LONG * p;
        register unsigned long l;
        int sw,sc,ew,ec;

        if (len==0) return;

        l=(c->Nl+(len<<3))&0xffffffffL;
        /* 95-05-24 eay Fixed a bug with the overflow handling, thanks to
         * Wei Dai <weidai@eskimo.com> for pointing it out. */
        if (l < c->Nl) /* переполнение */
                c->Nh++;
```

```
	c->Nh+=(len>>29);
	c->Nl=l;

	if (c->num != 0)
		{
		p=c->data;
		sw=c->num>>2;
		sc=c->num&0x03;

		if ((c->num+len) >= HASH_CBLOCK)
			{
			l=p[sw]; HOST_p_c2l(data,l,sc); p[sw++]=l;
			for (; sw<HASH_LBLOCK; sw++)
				{
				HOST_c2l(data,l); p[sw]=l;
				}
			HASH_BLOCK_HOST_ORDER (c,p,1);
			len-=(HASH_CBLOCK-c->num);
			c->num=0;
			/* drop through and do the rest */
			}
		else
			{
			c->num+=len;
			if ((sc+len) < 4) /* глупо добавлять символы к словам */
				{
				l=p[sw]; HOST_p_c2l_p(data,l,sc,len); p[sw]=l;
				}
			else
				{
				ew=(c->num>>2);
				ec=(c->num&0x03);
				l=p[sw]; HOST_p_c2l(data,l,sc); p[sw++]=l;
				for (; sw < ew; sw++)
					{
					HOST_c2l(data,l); p[sw]=l;
					}
				if (ec)
					{
					HOST_c2l_p(data,l,ec); p[sw]=l;
					}
				}
			return;
			}
		}

	sw=len/HASH_CBLOCK;
	if (sw > 0)
		{
#if defined(HASH_BLOCK_DATA_ORDER)
			{
			HASH_BLOCK_DATA_ORDER(c,data,sw);
			sw*=HASH_CBLOCK;
			data+=sw;
			len-=sw;
			}
#endif
		}

	if (len!=0)
		{
		p = c->data;
		c->num = len;
		ew=len>>2;	/* слов для копирования */
		ec=len&0x03;
		for (; ew; ew--,p++)
			{
```

```
                                HOST_c2l(data,l); *p=l;
                                }
                        HOST_c2l_p(data,l,ec);
                        *p=l;
                        }
        }
```

- обработка последнего блока и получение цифрового дайджеста

```
void MD4_Final(unsigned char *md, MD4_CTX *c)
        {
        register HASH_LONG *p;
        register unsigned long l;
        register int i,j;
        static const unsigned char end[4]={0x80,0x00,0x00,0x00};
        const unsigned char *cp=end;

        /* c->num определенно должен иметь место по крайней мере
     * еще для одного байта. */
        p=c->data;
        i=c->num>>2;
        j=c->num&0x03;

        l = (j==0) ? 0 : p[i];
        HOST_p_c2l(cp,l,j); p[i++]=l;
    /* i это следующее 'неопределенное слово' */

        if (i>(HASH_LBLOCK-2)) /* оставить место для Nl и Nh */
                {
                if (i<HASH_LBLOCK) p[i]=0;
                HASH_BLOCK_HOST_ORDER (c,p,1);
                i=0;
                }
        for (; i<(HASH_LBLOCK-2); i++)
                p[i]=0;

#if   defined(DATA_ORDER_IS_BIG_ENDIAN)
        p[HASH_LBLOCK-2]=c->Nh;
        p[HASH_LBLOCK-1]=c->Nl;
#elif defined(DATA_ORDER_IS_LITTLE_ENDIAN)
        p[HASH_LBLOCK-2]=c->Nl;
        p[HASH_LBLOCK-1]=c->Nh;
#endif
        HASH_BLOCK_HOST_ORDER (c,p,1);

#ifndef HASH_MAKE_STRING
#error "HASH_MAKE_STRING must be defined!"
#else
        HASH_MAKE_STRING(c,md);
#endif

        c->num=0;
        /* очистить все, HASH_BLOCK может оставить некоторую информацию
         * на стеке, но меня это не беспокоит
        memset((void *)c,0,sizeof(HASH_CTX));
         */
        }
```

На мой критический взгляд, очень длинно и очень запутано. Я постараюсь объяснить это тем, что библиотека OpenSSL создавалась как переносимая многоплатформенная библиотека, но... Пример реализации этих же самых примитивов приведенный в документе RFC 1320 намного проще, но не эффективнее.

Теперь мы готовы привести код для построения цифрового дайджеста строки:

```
MD4_CTX context;
unsigned char digest[16];
unsigned char string = "abc";
unsigned int len = strlen (string);

MD4_Init(&context);
MD4_Update(&context, string, len);
MD4_Final(digest, &context);
```

Просто, правда? Зато теперь мы знаем как это все работает внутри!

Ну и конечно же пример:

MD4("abc") = a448017aaf21d8525fc10ae87aa6729d

8. Построение цифрового дайджеста файла

Для того, чтобы построить цифровой дайджест любого файла, мы тоже можем использовать приведенные выше примитивы. Единственное отличие при обработке файла от обработки строки состоит в том, что большой файл мы не сможем представить в виде одной строки байт (символов) и поэтому его нужно будет обрабатывать частями, например вот так:

```
FILE *file;
MD4_CTX context;
int len;
unsigned char buffer[1024], digest[16];

file = fopen(filename, "rb");

MD4_Init(&context);
while (len = fread (buffer, 1, 1024, file))
   {
   MD4_Update(&context, buffer, len);
   }
MD4_Final(digest, &context);

fclose(file);
```

С помощью вот такого кусочка кода и был построен цифровой дайджест для файла WinWord.EXE, приведенный в начале статьи.

В связи с операцией построения цифрового дайджеста для файла возник один интересный вопрос, который я исследовал. Вопрос такой: а как влияет размер буфера данных на скорость работы алгоритма? Есть ли разница в том, какого размера буфер лучше использовать – размером в 1 байт или размером в 1000 байт?

Для этого я написал маленькую программку, которая измеряет скорость работы алгоритма для буфера различных размеров, начиная от размера в 1 байт и заканчивая размером в 128 байт. Результат работы программы я свел в следующие два графика – график скорости работы реализации и график пропускной способности реализации алгоритма. В измерениях я использовал две реализации алгоритма построения

цифрового дайджеста MD4 – реализацию, предложенную в документе RFC 1320 и реализацию из библиотеки OpenSSL.

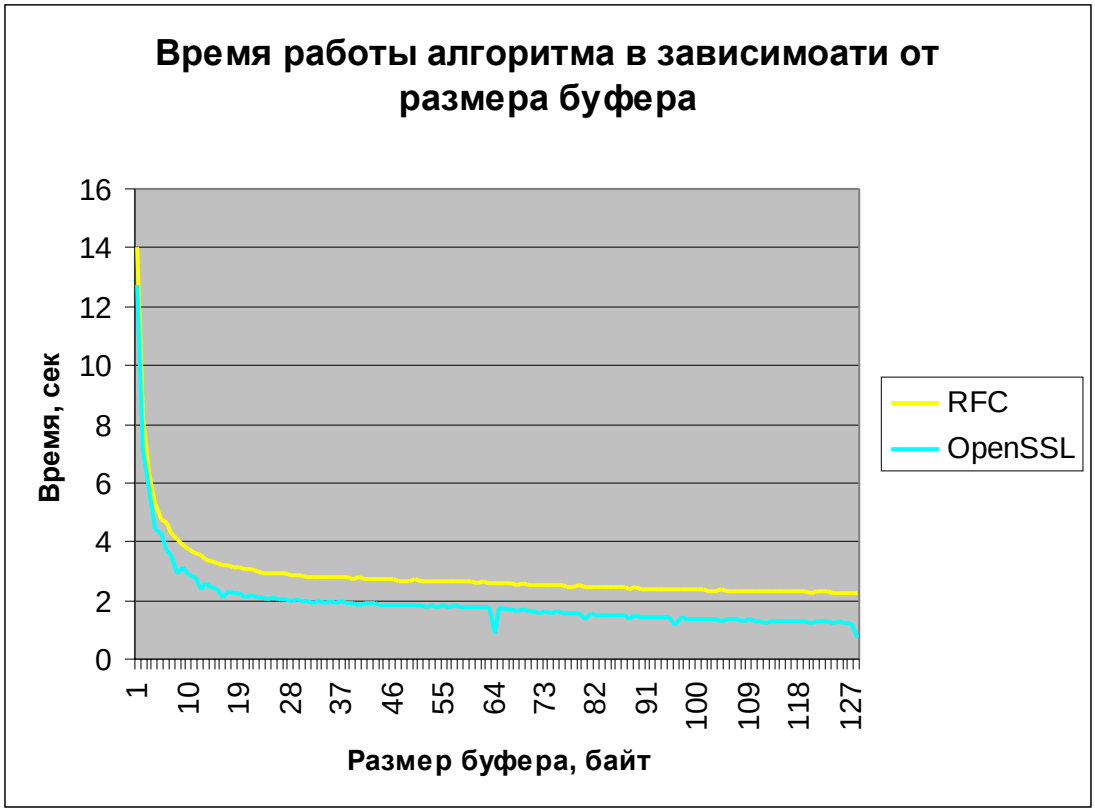

Рисунок 9. Скорость работы реализаций алгоритма MD4

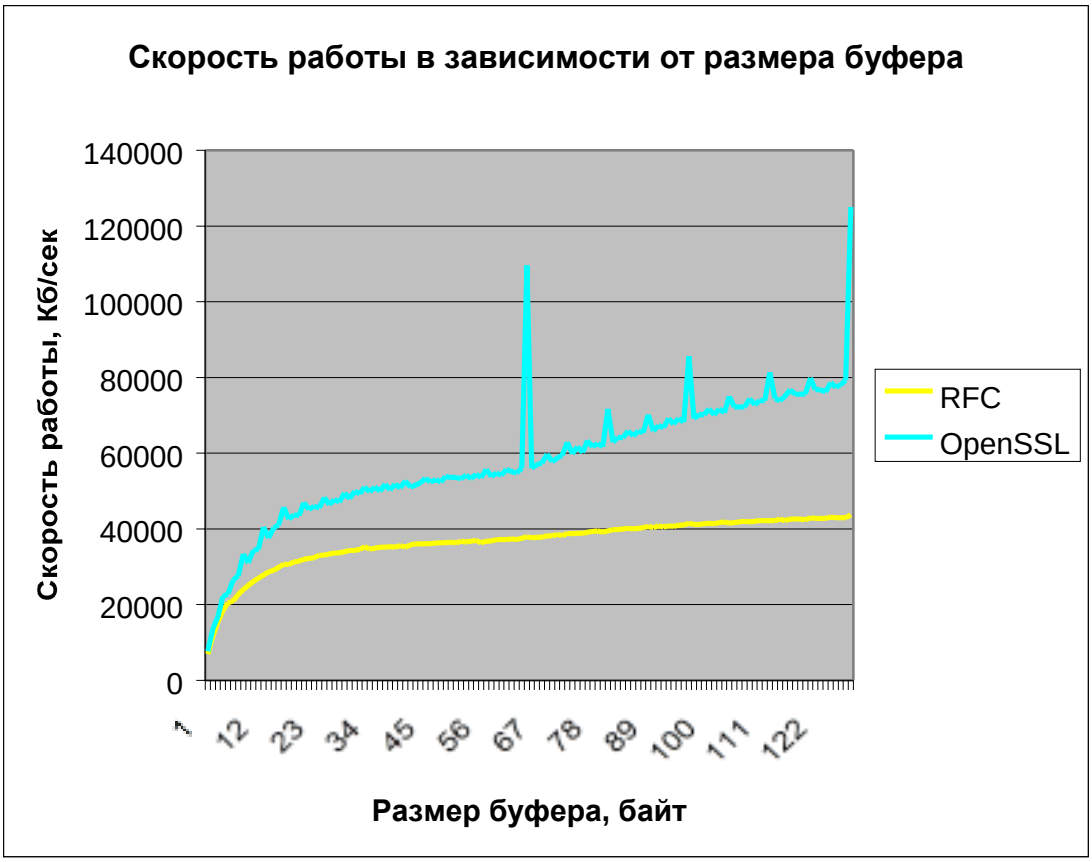

Рисунок 10. Пропускная способность реализаций алгоритма MD4

Измерения производились на компьютере с процессором Intel Pentium III, работающем на частоте 750 МГц. Оценка производилась на основании построения цифрового дайджеста 10000 раз для сообщения размером 10000 байт.

На основании этих двух графиков можно сделать следующие выводы:

- реализация алгоритма в библиотеке OpenSSL более эффективна;
- оптимальными для использования являются буферы с размерами кратными 64, то есть размеру внутреннего БЛОЧНОГО БУФЕРА;
- существуют некие другие размеры буфера, дающие увеличение пропускной способности;
- время работы реализации OpenSSL для буфера размером 64 байта составляет 0.891 секунды, для буфера размером 128 байт – 0.781 секунды;
- пропускная способность реализации OpenSSL для буфера размером 64 байта составляет 109602 килобайта в секунду (= 107 МБ/сек), для буфера размером 128 байт – 125040 килобайта в секунду (= 122 МБ/сек).

При использовании библиотеки OpenSSL выбирайте размер буфера кратный 64!

9. Резюме

Мы рассмотрели работу алгоритма построения цифрового дайджеста MD4. Этот алгоритм лежит в основе большинства основных методов построения цифровых дайджестов и теперь читателю будет несложно понять их работу. В следующей статье мы рассмотрим алгоритм построения цифрового дайджеста MD5, который широко используется сегодня. Мы коснемся математических вопросов для более глубокого понимания того, как и почему строится дайджест и почему его сложно "сломать". Также я постараюсь осветить вопросы криптоанализа и криптографических атак. До следующей встречи.

Редакция 3 от 24 сентября 2002 года
Ло Шу (LoShu_LoShu@Yahoo.COM, LoShu_LoShu@HotMail.COM).

Приложение A. RFC 1320

Network Working Group
Request for Comments: 1320
Obsoletes: RFC 1186

R. Rivest
MIT Laboratory for Computer Science
and RSA Data Security, Inc.
April 1992

Алгоритм построения цифрового дайджеста сообщения MD4

Статус данного документа

Данный документ предлагает информацию для рассмотрения сообществом Интернет. Документ не определяет стандарт Интернет. Распространение данного документа не ограничено.

Благодарности

Мы хотели бы выразить благодарность Дону Копперсмиту (Don Coppersmith), Бурту Калиски (Burt Kaliski), Ральфу Мерклу (Ralph Merkle) и Ноаму Нисану (Noam Nisan) за их многочисленные важные комментарии, замечания и предложения.

1. Краткое содержание

Этот документ описывает алгоритм построения цифрового дайджеста MD4 [1]. Алгоритм получает в качестве исходных данных сообщение произвольной длины и в результате своей работы формирует 128-битовый "отпечаток" или "цифровой дайджест" входного сообщения. Предполагается, что вычислительными средствами невозможно построить два сообщения, которые имеют одинаковые цифровые дайджесты, или построить сообщение с заданным цифровым дайджестом. Алгоритм MD4 предназначается для приложений, требующих использования цифровых подписей, в которых большие файлы должны быть "сжаты" безопасным способом перед тем как они будут зашифрованы приватным (секретным) ключом с использованием таких общественных криптосистем, как, например, RSA.

Алгоритм MD4 спроектирован таким образом, чтобы быть достаточно быстрым на 32-разрядных машинах. В дополнение к этому, алгоритм MD4 не требует никаких больших таблиц подстановки; алгоритм может быть достаточно компактно закодирован.

Алгоритм MD4 предложен для обзора общественности и возможной адаптации его в качестве стандарта.

Этот документ замещает RFC 1186 от октября 1990 года. Основное различие состоит в том, что реализация алгоритма MD4, приведенная в Приложении B, является более переносимой.

Для приложений, поддерживающих стандарт OSI, идентификатор объекта MD4 имеет вид

```
md4 OBJECT IDENTIFIER ::=
  {iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 4}
```

В типе AlgorithmIdentifier [3] из X.509, параметры для MD4 должны иметь тип NULL.

2. Терминология и нотация

В этом документе "слово" обозначает 32-разрядная величина, а "байт" это 8-разрядная величина. Последовательность бит может быть интерпретирована естественным образом как последовательность байт, где каждая последующая группа из восьми бит интерпретируется как байт, в котором старший (наиболее значимый) бит каждого байта стоит первым. Аналогично, последовательность байт может быть интерпретирована как последовательность 32-разрядных слов, где каждая последующая группа из четырех байт интерпретируется как слово, в котором младший (менее значимый) байт стоит первым.

Пусть x_I обозначает "x sub i". Если нижний индекс является выражением, то мы помещаем его в фигурные скобки, как, например, в x_{i+1}. Аналогично, мы используем ^ для верхних индексов (возведение в степень), таким образом, x^I обозначает x в степени i.

Пусть символ "+" обозначает сложение слов (то есть сложение по модулю 2^32). Пусть X <<< s обозначает 32-разрядную величину, полученную циклическим сдвигом (вращением) X влево на s разрядов. Пусть not(X) обозначает поразрядное дополнение X, и пусть X v Y обозначает поразрядное ИЛИ X т Y. Пусть X xor Y обозначает поразрядное исключающее ИЛИ X и Y, и пусть XY обозначает поразрядное И X и Y.

3. Описание алгоритма MD4

Мы начнем с предположения, что мы имеем b-разрядное сообщение в качестве входных данных и что мы хотим найти его цифровой дайджест. Здесь b произвольное неотрицательное число; b может быть равно нулю, оно не обязано быть кратно восьми и может быть произвольно большим. Представим, что разряды сообщения записаны следующим образом:

m_0 m_1 ... m_{b-1}

Чтобы построить цифровой дайджест сообщения необходимо выполнить следующие пять шагов.

3.1 Шаг 1. Расширение сообщения

Сообщение расширяется таким образом, чтобы его длина (в битах) была конгруэнтна 448 по модулю 512. То есть сообщение расширяется до размера так, что ему не достает всего 64 бита, чтобы иметь длину кратную 512. Расширение сообщения производится всегда, даже если его длина уже конгруэнтна 448 по модулю 512.

Расширение сообщения выполняется следующим образом: единственный бит "1" добавляется в конец сообщения, а затем сообщение дополняется таким количеством "0" бит, чтобы его длина стала конгруэнтна 448 по модулю 512. В целом, к сообщению будет добавлено от одного до 512 бит.

3.2 Шаг 2. Добавление длины

64-разрядное представление величины b (то есть длины сообщения до того как оно было расширено) добавляется в конец результата, полученного на предыдущем шаге. В том случае, если величина b больше чем 2^64, то добавляются только младшие 64 бита величины b. (Эти биты добавляются как два 32-разрядных слова и добавляется вначале младшее слово в соответствии с предыдущими соглашениями.)

К этому моменту результирующее сообщение (после расширения и добавления величины b) имеет длину, которая в точности кратна 512 битам. Эквивалентно, это сообщение имеет длину, которая в точности кратна 16 (32-разрядным) словам. Пусть M[0 ... N-1] обозначает слова результирующего сообщения, где N в точности кратно 16.

3.3 Шаг 3. Инициализация MD буфера

Для вычисления цифрового дайджеста используется буфер размером в четыре слова (A, B, C, D). Здесь каждое из A, B, C, D представляет собой 32-разрядный регистр. Эти регистры инициализируются следующими шестнадцатеричными величинами (младший байт стоит первым):

слово A: 01 23 45 67
слово B: 89 ab cd ef
слово C: fe dc ba 98
слово D: 76 54 32 10

3.4 Шаг 4. Обработать сообщение блоками по 16 слов

Сперва мы определим три вспомогательные функции, каждая из которых имеет три 32-разряных слова в качестве входных данных и в качестве результата формирует одно 32-разрядное слово.

F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XY v XZ v YZ
H(X,Y,Z) = X xor Y xor Z

В каждом разряде F действует как условие: если X, то Y, иначе Z. Функция F может быть определена с использованием операции "+" вместо операции "v", так как XY and not(X)Z никогда не будет иметь "1" бит в одной и той же позиции. (???) В каждом разряде G действует как функция большинства (majority): если по крайней мере два из X, Y, Z имеют "1", то G поместит "1" в это разряд, иначе G поместит "0". Интересно отметить, что если биты X, Y и Z независимы и на них ничто не влияет, то и каждый бит F(X,Y,Z) будет также независим. Функция H представляет собой поразрядное исключающее ИЛИ, или функцию "четности"; она обладает такими же свойствами как и функции F и G.

Выполнить следующее:

```
/* Обработать каждый блок из 16 слов. */
For i = 0 to N/16-1 do

  /* Копировать блок i в X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* цикла по j */

  /* Сохранить A как AA, B как BB, C как CC, и D как DD. */
  AA = A
  BB = B
  CC = C
  DD = D
```

```
/* Раунд 1. */
/* Пусть [abcd k s] обозначает операцию
     a = (a + F(b,c,d) + X[k]) <<< s. */
/* Выполнить следующие 16 операций. */
[ABCD  0  3]  [DABC  1  7]  [CDAB  2 11]  [BCDA  3 19]
[ABCD  4  3]  [DABC  5  7]  [CDAB  6 11]  [BCDA  7 19]
[ABCD  8  3]  [DABC  9  7]  [CDAB 10 11]  [BCDA 11 19]
[ABCD 12  3]  [DABC 13  7]  [CDAB 14 11]  [BCDA 15 19]

/* Раунд 2. */
/* Пусть [abcd k s] обозначает операцию
     a = (a + G(b,c,d) + X[k] + 5A827999) <<< s. */
/* Выполнить следующие 16 операций. */
[ABCD  0  3]  [DABC  4  5]  [CDAB  8  9]  [BCDA 12 13]
[ABCD  1  3]  [DABC  5  5]  [CDAB  9  9]  [BCDA 13 13]
[ABCD  2  3]  [DABC  6  5]  [CDAB 10  9]  [BCDA 14 13]
[ABCD  3  3]  [DABC  7  5]  [CDAB 11  9]  [BCDA 15 13]

/* Раунд 3. */
/* Пусть [abcd k s] обозначает операцию
     a = (a + H(b,c,d) + X[k] + 6ED9EBA1) <<< s. */
/* Выполнить следующие 16 операций. */
[ABCD  0  3]  [DABC  8  9]  [CDAB  4 11]  [BCDA 12 15]
[ABCD  2  3]  [DABC 10  9]  [CDAB  6 11]  [BCDA 14 15]
[ABCD  1  3]  [DABC  9  9]  [CDAB  5 11]  [BCDA 13 15]
[ABCD  3  3]  [DABC 11  9]  [CDAB  7 11]  [BCDA 15 15]

/* Затем выполнить следующие сложения. (То есть нарастить каждый
   из четырех регистров на величину, которую он содержал перед началом
   обработки этого блока.) */
A = A + AA
B = B + BB
C = C + CC
D = D + DD

end /* цикла по i */
```

Примечание. Величина 5A..99 представляет собой шестнадцатеричную 32-разрядную константу, записанную начиная со старшей цифры. Эта константа представляет квадратный корень из 2. Восьмеричное значение этой константы 013240474631.

Величина 6E..A1 представляет собой шестнадцатеричную 32-разрядную константу, записанную начиная со старшей цифры. Эта константа представляет квадратный корень из 3. Восьмеричное значение этой константы 015666365641.

Смотрите: Кнут, Искусство программирования ЭВМ, том 2 (получисленные алгоритмы), Второе издание (1981), Addison-Wesley. Таблица 2, страница 660.

3.5 Шаг 5. Вывод результата

Цифровой дайджест сообщения представляет собой результат вывода содержимого регистров A, B, C, D. То есть мы начинаем с младшего байта A и заканчиваем старшим байтом D.

Это завершает описание алгоритма MD4. Пример реализации на языке программирования Си приведен в Приложении B.

4. Резюме

Алгоритм построения цифрового дайджеста сообщения MD4 прост в реализации и обеспечивает построение "отпечатка" или дайджеста сообщения для сообщения произвольной длины. Предполагается, что сложность построения двух сообщений с одинаковым дайджестом имеет порядок 2^64 операций, и что сложность построения сообщения по заданному цифровому дайджесту имеет порядок 2^128 операций. Алгоритм MD4 был очень внимательно и тщательно исследован на предмет наличия слабостей. Однако, данный алгоритм является достаточно новым алгоритмом и дальнейший криптографический анализ оправдан наряду с различными предложениями на этот счет.

Литература

[1] Rivest, R., "The MD4 message digest algorithm", in A.J. Menezes and S.A. Vanstone, editors, Advances in Cryptology - CRYPTO '90 Proceedings, pages 303-311, Springer-Verlag, 1991.

[2] Rivest, R., "The MD4 Message Digest Algorithm", RFC 1186, MIT, October 1990.

[3] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework".

[4] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, MIT and RSA Data Security, Inc, April 1992.

ПРИЛОЖЕНИЕ B – Пример реализации

Это приложение содержит следующие файлы:

global.h – глобальный заголовочный файл

md4.h – заголовочный файл для MD4

md4c.c – исходный код для MD4

mddriver.c – тестовая программа для MD2, MD4 и MD5

Тестовая программа по умолчанию компилируется для алгоритма MD5, но может также быть скомпилирована для алгоритмов MD2 и MD4, если символ MD определен в строке компилятора Си как 2 или 4.

Реализация переносима и должна работать на различных платформах. Однако, несложно оптимизировать эту реализацию для специфических платформ, и остается в качестве упражнения для читателя. Например, на платформах с "обратным следованием байт" ("little-endian"), где байт с младшим адресом в 32-разрядном слове является наименее значимым и нет ограничений на выравнивание, вызов Decode в MD4Transform может быть заменен простым изменением типа.

A.1 global.h

```
/* GLOBAL.H - RSAREF типы и константы
 */

/* PROTOTYPES должен быть определен как 1 если и только если компилятор
```

```
    поддерживает прототипы аргументов функций.
  Следующая строка определяет PROTOTYPES как 0 если он не был ранее
    определен через параметры командной строки компилятора Си.
 */
#ifndef PROTOTYPES
#define PROTOTYPES 0
#endif

/* POINTER определяет универсальный тип указателя */
typedef unsigned char *POINTER;

/* UINT2 определяет двух-байтовое слово */
typedef unsigned short int UINT2;

/* UINT4 определяет четырех-байтовое слово */
typedef unsigned long int UINT4;

/* PROTO_LIST определяется на основании того, как ранее был определен
PROTOTYPES.
  Если PROTOTYPES используется, то PROTO_LIST возвращает список, в
противном
    Случае он возвращает пустой список.
 */

#if PROTOTYPES
#define PROTO_LIST(list) list
#else
#define PROTO_LIST(list) ()
#endif
```

A.2 md4.h

```
/* MD4.H – заголовочный фал для MD4C.C
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
  rights reserved.

  License to copy and use this software is granted provided that it
  is identified as the "RSA Data Security, Inc. MD4 Message-Digest
  Algorithm" in all material mentioning or referencing this software
  or this function.

  License is also granted to make and use derivative works provided
  that such works are identified as "derived from the RSA Data
  Security, Inc. MD4 Message-Digest Algorithm" in all material
  mentioning or referencing the derived work.

  RSA Data Security, Inc. makes no representations concerning either
  the merchantability of this software or the suitability of this
  software for any particular purpose. It is provided "as is"
  without express or implied warranty of any kind.

  These notices must be retained in any copies of any part of this
  documentation and/or software.
 */

/* Контест MD4. */
typedef struct {
  UINT4 state[4];                    /* состояние (ABCD) */
```

```
  UINT4 count[2];      /* количество бит по модулю 2^64 (lsb first) */
  unsigned char buffer[64];                    /* буфер ввода */
} MD4_CTX;

void MD4Init PROTO_LIST ((MD4_CTX *));
void MD4Update PROTO_LIST
  ((MD4_CTX *, unsigned char *, unsigned int));
void MD4Final PROTO_LIST ((unsigned char [16], MD4_CTX *));
```

A.3 md4c.c

```
/* MD4C.C - RSA Data Security, Inc., Алгоритм построения цифрового дайджеста
   сообщения MD4.
 */

/* Copyright (C) 1990-2, RSA Data Security, Inc. All rights reserved.

   License to copy and use this software is granted provided that it
   is identified as the "RSA Data Security, Inc. MD4 Message-Digest
   Algorithm" in all material mentioning or referencing this software
   or this function.

   License is also granted to make and use derivative works provided
   that such works are identified as "derived from the RSA Data
   Security, Inc. MD4 Message-Digest Algorithm" in all material
   mentioning or referencing the derived work.

   RSA Data Security, Inc. makes no representations concerning either
   the merchantability of this software or the suitability of this
   software for any particular purpose. It is provided "as is"
   without express or implied warranty of any kind.

   These notices must be retained in any copies of any part of this
   documentation and/or software.
 */

#include "global.h"
#include "md4.h"

/* Константы для процедуры MD4Transform.
 */
#define S11 3
#define S12 7
#define S13 11
#define S14 19
#define S21 3
#define S22 5
#define S23 9
#define S24 13
#define S31 3
#define S32 9
#define S33 11
#define S34 15

static void MD4Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST
  ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST
  ((UINT4 *, unsigned char *, unsigned int));
static void MD4_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
```

```
static void MD4_memset PROTO_LIST ((POINTER, int, unsigned int));

static unsigned char PADDING[64] = {
  0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G и H – основные функции MD4.
 */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (y)) | ((x) & (z)) | ((y) & (z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))

/* ROTATE_LEFT вращает x влево на n разрядов.
 */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG и HH это преобразования для раундов 1, 2 и 3 */
/* Вращение реализовано отдельно от сложения, чтобы предотвратить
 повторное вычисление */

#define FF(a, b, c, d, x, s) { \
    (a) += F ((b), (c), (d)) + (x); \
    (a) = ROTATE_LEFT ((a), (s)); \
  }
#define GG(a, b, c, d, x, s) { \
    (a) += G ((b), (c), (d)) + (x) + (UINT4)0x5a827999; \
    (a) = ROTATE_LEFT ((a), (s)); \
  }
#define HH(a, b, c, d, x, s) { \
    (a) += H ((b), (c), (d)) + (x) + (UINT4)0x6ed9eba1; \
    (a) = ROTATE_LEFT ((a), (s)); \
  }

/* Инициализация MD4. Начинает операцию MD4, создавая новый контекст.
 */
void MD4Init (context)
MD4_CTX *context;                                       /* контекст */
{
  context->count[0] = context->count[1] = 0;

  /* Загрузить "магические" начальные константы.
   */
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
}

/* Операция MD4 обновления блока. Продолжает операцию построения
цифрового
 дайджеста сообщения MD4, обрабатывая очередной блок сообщения,
 и обновляя контекст.
*/
void MD4Update (context, input, inputLen)
MD4_CTX *context;                                       /* контекст */
unsigned char *input;                                   /* входной блок */
unsigned int inputLen;                                  /* длина входного блока */
{
```

```
  unsigned int i, index, partLen;

  /* Вычислить число байт по модулю 64 */
  index = (unsigned int)((context->count[0] >> 3) & 0x3F);

  /* Обновить число бит */
  if ((context->count[0] += ((UINT4)inputLen << 3))
      < ((UINT4)inputLen << 3))
    context->count[1]++;
  context->count[1] += ((UINT4)inputLen >> 29);

  partLen = 64 - index;

  /* Выполнить преобразование столько раз сколь это необходимо.
   */
  if (inputLen >= partLen) {
    MD4_memcpy
      ((POINTER)&context->buffer[index], (POINTER)input, partLen);
    MD4Transform (context->state, context->buffer);

    for (i = partLen; i + 63 < inputLen; i += 64)
      MD4Transform (context->state, &input[i]);

    index = 0;
  }
  else
    i = 0;

  /* Поместить остаток входных данных в буфер */
  MD4_memcpy
    ((POINTER)&context->buffer[index], (POINTER)&input[i],
     inputLen-i);
}

/* Окончание работы MD4. Завершает операцию построения цифрового
дайджеста
сообщения MD4, записывая дайджест сообщения и обнуляя контекст.
 */
void MD4Final (digest, context)
unsigned char digest[16];                   /* дайджест сообщения */
MD4_CTX *context;                               /* контекст */
{
  unsigned char bits[8];
  unsigned int index, padLen;

  /* Сохранить число бит */
  Encode (bits, context->count, 8);

  /* Расширить до 56 mod 64.
   */
  index = (unsigned int)((context->count[0] >> 3) & 0x3f);
  padLen = (index < 56) ? (56 - index) : (120 - index);
  MD4Update (context, PADDING, padLen);

  /* Добавить длину сообщения (до расширения) */
  MD4Update (context, bits, 8);
  /* Сохранить состояние в дайджест */
  Encode (digest, context->state, 16);

  /* Обнулить контекст.
```

```
  */
  MD4_memset ((POINTER)context, 0, sizeof (*context));

}

/* Основное преобразование MD4. Преобразует state основываясь на block.
 */
static void MD4Transform (state, block)
UINT4 state[4];
unsigned char block[64];
{
  UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

  Decode (x, block, 64);

  /* Раунд 1 */
  FF (a, b, c, d, x[ 0], S11); /* 1 */
  FF (d, a, b, c, x[ 1], S12); /* 2 */
  FF (c, d, a, b, x[ 2], S13); /* 3 */
  FF (b, c, d, a, x[ 3], S14); /* 4 */
  FF (a, b, c, d, x[ 4], S11); /* 5 */
  FF (d, a, b, c, x[ 5], S12); /* 6 */
  FF (c, d, a, b, x[ 6], S13); /* 7 */
  FF (b, c, d, a, x[ 7], S14); /* 8 */
  FF (a, b, c, d, x[ 8], S11); /* 9 */
  FF (d, a, b, c, x[ 9], S12); /* 10 */
  FF (c, d, a, b, x[10], S13); /* 11 */
  FF (b, c, d, a, x[11], S14); /* 12 */
  FF (a, b, c, d, x[12], S11); /* 13 */
  FF (d, a, b, c, x[13], S12); /* 14 */
  FF (c, d, a, b, x[14], S13); /* 15 */
  FF (b, c, d, a, x[15], S14); /* 16 */

  /* Раунд 2 */
  GG (a, b, c, d, x[ 0], S21); /* 17 */
  GG (d, a, b, c, x[ 4], S22); /* 18 */
  GG (c, d, a, b, x[ 8], S23); /* 19 */
  GG (b, c, d, a, x[12], S24); /* 20 */
  GG (a, b, c, d, x[ 1], S21); /* 21 */
  GG (d, a, b, c, x[ 5], S22); /* 22 */
  GG (c, d, a, b, x[ 9], S23); /* 23 */
  GG (b, c, d, a, x[13], S24); /* 24 */
  GG (a, b, c, d, x[ 2], S21); /* 25 */
  GG (d, a, b, c, x[ 6], S22); /* 26 */
  GG (c, d, a, b, x[10], S23); /* 27 */
  GG (b, c, d, a, x[14], S24); /* 28 */
  GG (a, b, c, d, x[ 3], S21); /* 29 */
  GG (d, a, b, c, x[ 7], S22); /* 30 */
  GG (c, d, a, b, x[11], S23); /* 31 */
  GG (b, c, d, a, x[15], S24); /* 32 */

  /* Раунд 3 */
  HH (a, b, c, d, x[ 0], S31); /* 33 */
  HH (d, a, b, c, x[ 8], S32); /* 34 */
  HH (c, d, a, b, x[ 4], S33); /* 35 */
  HH (b, c, d, a, x[12], S34); /* 36 */
  HH (a, b, c, d, x[ 2], S31); /* 37 */
  HH (d, a, b, c, x[10], S32); /* 38 */
  HH (c, d, a, b, x[ 6], S33); /* 39 */
  HH (b, c, d, a, x[14], S34); /* 40 */
```

```
  HH (a, b, c, d, x[ 1], S31); /* 41 */
  HH (d, a, b, c, x[ 9], S32); /* 42 */
  HH (c, d, a, b, x[ 5], S33); /* 43 */
  HH (b, c, d, a, x[13], S34); /* 44 */
  HH (a, b, c, d, x[ 3], S31); /* 45 */
  HH (d, a, b, c, x[11], S32); /* 46 */
  HH (c, d, a, b, x[ 7], S33); /* 47 */
  HH (b, c, d, a, x[15], S34); /* 48 */

  state[0] += a;
  state[1] += b;
  state[2] += c;
  state[3] += d;

  /* Заполнить нулями область с важной информацией.
   */
  MD4_memset ((POINTER)x, 0, sizeof (x));
}

/* Кодирует input (UINT4) в output (unsigned char). Предполагается, что len
 кратна 4.
 */
static void Encode (output, input, len)
unsigned char *output;
UINT4 *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4) {
    output[j] = (unsigned char)(input[i] & 0xff);
    output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
    output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
    output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
  }
}

/* Декодирует input (unsigned char) в output (UINT4). Предполагается, что len
 кратна 4.
*/
static void Decode (output, input, len)

UINT4 *output;
unsigned char *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4)
    output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) |
      (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}

/* Примечание: Замените "for loop" стандартной функцией memcpy если
 это возможно.
 */
static void MD4_memcpy (output, input, len)
POINTER output;
POINTER input;
unsigned int len;
```

```
{
  unsigned int i;

  for (i = 0; i < len; i++)
    output[i] = input[i];
}

/* Примечание: Замените "for loop" стандартной функцией memset если
 это возможно.
 */
static void MD4_memset (output, value, len)
POINTER output;
int value;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
    ((char *)output)[i] = (char)value;
}
```

A.4 mddriver.c

```
/* MDDRIVER.C – тестовая программа для MD2, MD4 и MD5
 */

/* Copyright (C) 1990-2, RSA Data Security, Inc. Created 1990. All
   rights reserved.

   RSA Data Security, Inc. makes no representations concerning either
   the merchantability of this software or the suitability of this
   software for any particular purpose. It is provided "as is"
   without express or implied warranty of any kind.

   These notices must be retained in any copies of any part of this
   documentation and/or software.

 */

/* Следующая строка определяет MD как MD5 по умолчанию, если это не было
 определено через флаг Си компилятора.
 */
#ifndef MD
#define MD MD5
#endif

#include <stdio.h>
#include <time.h>
#include <string.h>
#include "global.h"
#if MD == 2
#include "md2.h"
#endif
#if MD == 4
#include "md4.h"
#endif
#if MD == 5
#include "md5.h"
#endif
```

```
/* Длина тестового блока, число тестовых блоков.
 */
#define TEST_BLOCK_LEN 1000
#define TEST_BLOCK_COUNT 1000

static void MDString PROTO_LIST ((char *));
static void MDTimeTrial PROTO_LIST ((void));
static void MDTestSuite PROTO_LIST ((void));
static void MDFile PROTO_LIST ((char *));
static void MDFilter PROTO_LIST ((void));
static void MDPrint PROTO_LIST ((unsigned char [16]));

#if MD == 2
#define MD_CTX MD2_CTX
#define MDInit MD2Init
#define MDUpdate MD2Update
#define MDFinal MD2Final
#endif
#if MD == 4
#define MD_CTX MD4_CTX
#define MDInit MD4Init
#define MDUpdate MD4Update
#define MDFinal MD4Final
#endif
#if MD == 5
#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final
#endif

/* Главная функция.

   Аргументы (может присутствовать любая комбинация):
    -sstring – построить дайджест строки
    -t      - выполнить временной тест
    -x      - построить дайджест для тестового набора
    filename – построить дайджест файла
    (none)   - построить дайджест стандартного входного потока
 */
int main (argc, argv)
int argc;
char *argv[];
{
  int i;

  if (argc > 1)
    for (i = 1; i < argc; i++)
      if (argv[i][0] == '-' && argv[i][1] == 's')
        MDString (argv[i] + 2);
      else if (strcmp (argv[i], "-t") == 0)
        MDTimeTrial ();
      else if (strcmp (argv[i], "-x") == 0)
        MDTestSuite ();
      else
        MDFile (argv[i]);
  else
    MDFilter ();

  return (0);
```

```
}

/* Построить дайджест строки и вывести результат.
 */
static void MDString (string)
char *string;
{
  MD_CTX context;
  unsigned char digest[16];
  unsigned int len = strlen (string);

  MDInit (&context);
  MDUpdate (&context, string, len);
  MDFinal (digest, &context);

  printf ("MD%d (\"%s\") = ", MD, string);
  MDPrint (digest);
  printf ("\n");
}

/* Измерить время построения дайджеста для TEST_BLOCK_COUNT блоков
длиной
 TEST_BLOCK_LEN.
 */
static void MDTimeTrial ()
{
  MD_CTX context;
  time_t endTime, startTime;
  unsigned char block[TEST_BLOCK_LEN], digest[16];
  unsigned int i;

  printf
    ("MD%d time trial. Digesting %d %d-byte blocks ...", MD,
     TEST_BLOCK_LEN, TEST_BLOCK_COUNT);

  /* Инициализировать блок */
  for (i = 0; i < TEST_BLOCK_LEN; i++)
    block[i] = (unsigned char)(i & 0xff);

  /* Запустить таймер */
  time (&startTime);

  /* Построить дайджест блоков */
  MDInit (&context);
  for (i = 0; i < TEST_BLOCK_COUNT; i++)
    MDUpdate (&context, block, TEST_BLOCK_LEN);
  MDFinal (digest, &context);

  /* Остановить таймер */
  time (&endTime);

  printf (" done\n");
  printf ("Digest = ");
  MDPrint (digest);
  printf ("\nTime = %ld seconds\n", (long)(endTime-startTime));
  printf
    ("Speed = %ld bytes/second\n",
     (long)TEST_BLOCK_LEN * (long)TEST_BLOCK_COUNT/(endTime-startTime));
}
```

```
/* Построить дайджест для тестового набора и вывести результат.
 */
static void MDTestSuite ()
{
  printf ("MD%d test suite:\n", MD);

  MDString ("");
  MDString ("a");
  MDString ("abc");
  MDString ("message digest");
  MDString ("abcdefghijklmnopqrstuvwxyz");
  MDString
    ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");
  MDString

    ("12345678901234567890123456789012345678901\
23456789012345678901234567890123456789012345678901234567890");
}

/* Построить дайджест файла и вывести результат.
 */
static void MDFile (filename)
char *filename;
{
  FILE *file;
  MD_CTX context;
  int len;
  unsigned char buffer[1024], digest[16];

  if ((file = fopen (filename, "rb")) == NULL)
    printf ("%s can't be opened\n", filename);

  else {
    MDInit (&context);
    while (len = fread (buffer, 1, 1024, file))
      MDUpdate (&context, buffer, len);
    MDFinal (digest, &context);

    fclose (file);

    printf ("MD%d (%s) = ", MD, filename);
    MDPrint (digest);
    printf ("\n");
  }
}

/* Построить дайджест стандартного потока ввода и вывести результат.
 */
static void MDFilter ()
{
  MD_CTX context;
  int len;
  unsigned char buffer[16], digest[16];

  MDInit (&context);
  while (len = fread (buffer, 1, 16, stdin))
    MDUpdate (&context, buffer, len);
  MDFinal (digest, &context);

  MDPrint (digest);
```

```
  printf ("\n");
}

/* Печатает дайджест сообщения в шестнадцатеричном виде.
 */
static void MDPrint (digest)
unsigned char digest[16];

{
  unsigned int i;

  for (i = 0; i < 16; i++)
    printf ("%02x", digest[i]);
}
```

A.5 Набор тестов

Набор тестов для MD4 (опция тестовой программы "-x") должен печатать следующие результаты:

```
MD4 test suite:
MD4 ("") = 31d6cfe0d16ae931b73c59d7e0c089c0
MD4 ("a") = bde52cb31de33e46245e05fbdbd6fb24
MD4 ("abc") = a448017aaf21d8525fc10ae87aa6729d
MD4 ("message digest") = d9130a8164549fe818874806e1c7014b
MD4 ("abcdefghijklmnopqrstuvwxyz") = d79e1c308aa5bbcdeea8ed63df412da9
MD4 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
043f8582f241db351ce627e153e7f0e4
MD4 ("12345678901234567890123456789012345678901234567890123456789012345678
78901234567890") = e33b4ddc9c38f2199c3e7b164fcc0536
```

Соглашения по безопасности

Уровень безопасности, обсуждаемый в данной статье, предполагается достаточным для реализации гибридной схемы цифровой подписи средней степени защиты, основанной на MD4 и криптосистеме с открытым ключом. Мы не знаем ни одной причины, по которой MD4 был бы недостаточным для реализации схем цифровой подписи с сильной степенью защиты, но так как MD4 был спроектирован чтобы быть исключительно быстрым алгоритмом, он стоит на грани риска от успешных криптографических атак. После последующего критического анализа MD4 может быть рекомендован для приложений с очень высокой степенью защиты. Для приложений с очень высокой степенью защиты, перед завершением этого обзора, рекомендуется алгоритм MD5 [4].

Адрес автора

Ronald L. Rivest
Massachusetts Institute of Technology
Laboratory for Computer Science
NE43-324
545 Technology Square
Cambridge, MA 02139-1986

Phone: (617) 253-5880
EMail: rivest@theory.lcs.mit.edu