Алгоритм построения цифрового дайджеста MD5

Введение

Данная статья продолжает рассмотрение алгоритмов построения цифрового дайджеста сообщения и содержит информацию об алгоритме MD5. По-прежнему, предполагается, что читатель знаком с языками программирования Си и Форт. Смотрите статью "Алгоритм построения цифрового дайджеста MD4" в предыдущем номере журнала.

Цифровой дайджест MD5

В том же 1991 году, когда был создан алгоритм построения цифрового дайджеста MD4, профессором Ривестом был предложен другой алгоритм – MD5, "старший" и более сильный брат алгоритма MD4 (почему "старший" мы узнаем чуть позже). Он во многом похож на своего собрата и поэтому нам будет гораздо легче понять как он работает, так как мы уже в деталях знаем как работает MD4.

Здесь я тоже начну с примера. Итак, цифровой дайджест строки символов и файла:

MD5("MD5") = 7F138A09169B250E9DCB378140907378
MD5(WinWord.EXE) = B9D86A2E86028000B500B498FE71C299

Длина цифрового дайджеста MD5 составляет 16 байт или 16*8 = 128 бит.

MD5 процессор

MD5 процессор состоит из набора следующих регистров. Прежде всего это регистры A, B, C и D, которые содержат промежуточный результат вычисления цифрового дайджеста. "БЛОЧНЫЙ БУФЕР" MD5 процессора содержит 16 32-разрядных слов, в этом буфере формируется очередной блок для построения очередного промежуточного результата. Регистр "БАЙТ В БУФЕРЕ" содержит число, определяющее количество байт в "БЛОЧНОМ БУФЕРЕ" на этапе его формирования, или заполнения. Регистр "РАЗМЕР СООБЩЕНИЯ" содержит полный размер исходного сообщения, для которого строиться цифровой дайджест, в битах; эта величина будет добавлена в конец исходного сообщения на финальном этапе формирования цифрового дайджеста. Регистры "УКАЗАТЕЛЬ ДАННЫХ" и "РАЗМЕР ДАННЫХ" содержат, соответственно, указатель на текущий обрабатываемый пакет данных и его размер.

Структура MD5 процессора изображена графически на Рисунке 1.

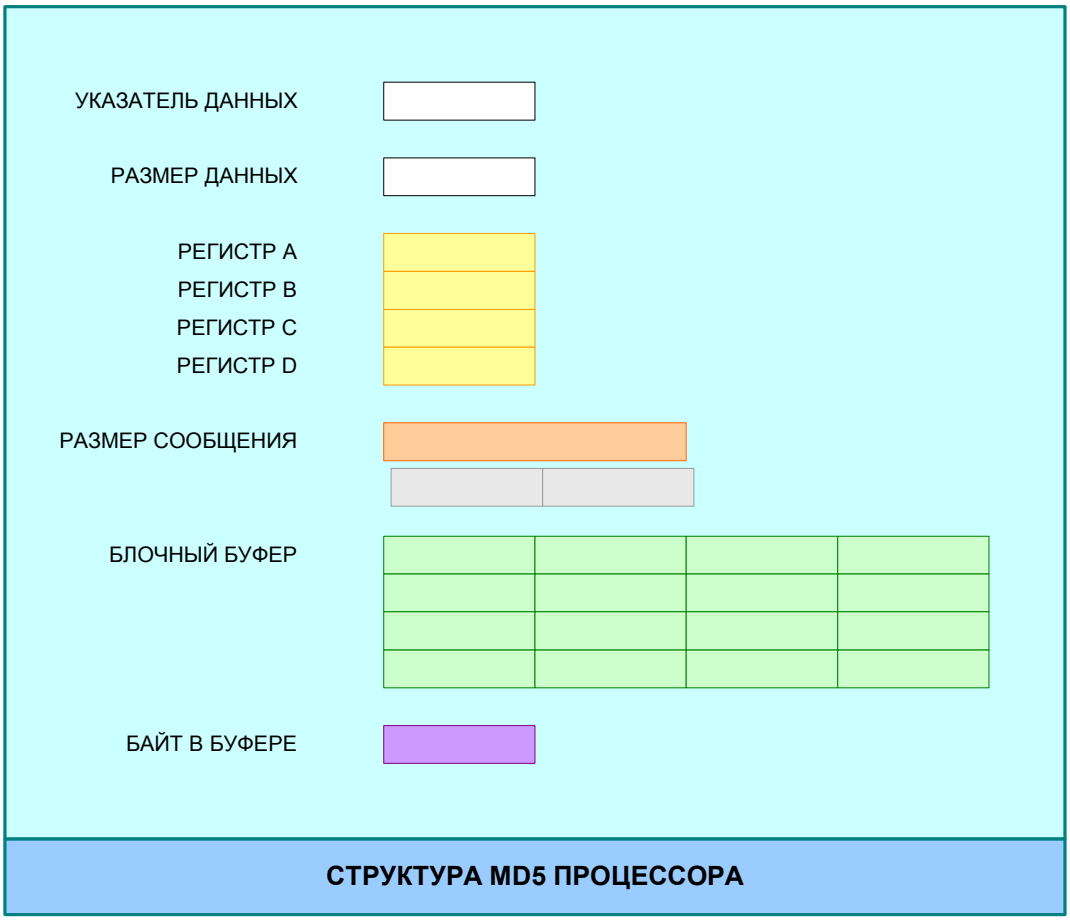


Рисунок 1. Структура MD5 процессора.

Операция инициализации процессора приводи его в определенное начальное состояние, то есть во все регистры помещаются определенные начальные значения. Эти значения диктуются документом RFC 1321. После инициализации MD5 процессор выглядит следующим образом:

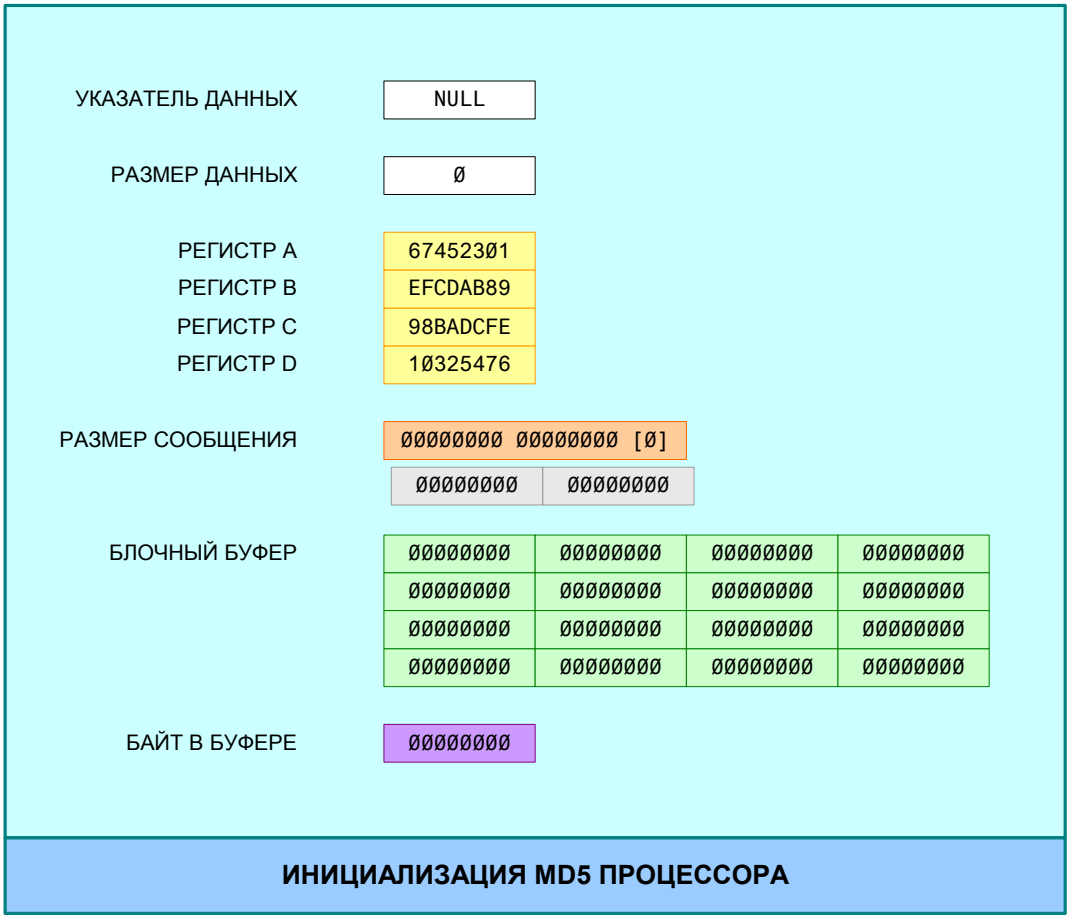


Рисунок 3. Состояние MD5 процессора после инициализации.

В качестве маленького итога, можно сказать, что MD5 процессор по своей структуре не отличается от MD4 процессора. Они даже сходны по своей процедуре инициализации.

Базовые операции

Цифровой дайджест MD5 определяет четыре базовые логические функции, которые используются в операции преобразования блока данных. Вот они:

F(x,y,z) = (x AND y) OR (NOT x AND z)[1]
G(x,y,z) = (x AND z) OR (y AND NOT z)[2]
H(x,y,z) = x XOR y XOR z
I(x,y,z) = y XOR (x OR NOT z)[3]

И таблицы истинности этих функций:

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

x	y	Z	G
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**0**

X	y	z	H
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**

x	y	z	I
0	0	0	**1**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**

[1] NOT x AND z = (NOT x) AND z
[2] y AND NOT z = y AND (NOT z)
[3] x OR NOT z = x OR (NOT z)

1	0	1	**0**
1	1	0	**1**
1	1	1	**1**

1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

1	0	1	**0**
1	1	0	**0**
1	1	1	**1**

1	0	1	**1**
1	1	0	**0**
1	1	1	**0**

На языке программирования Си эти логические функции могут быть записаны следующим образом:

```
#define  F(x,y,z)  ((x & y) | (~x & z))
#define  G(x,y,z)  ((x & z) | (y & ~z))
#define  H(x,y,z)  (x ^ y ^ z)
#define  I(x,y,z)  (y ^ (x | ~z))
```

Библиотека OpenSSL применяет несколько другие обозначения и использует оптимизацию! Логические функции можно преобразовать и уменьшить количество операций, требуемых для их вычисления:

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

Здесь используется тот факт, что (x AND y) OR (NOT x AND z) = ((y XOR z) AND z) XOR z. То есть мы имеем на одну логическую операцию меньше. Сравните: в первом случае – AND, OR, NOT, AND, и во втором случае – XOR, AND, XOR. Как увидим дальше, экономия в одну операцию будет существенной во время преобразования блока данных.

Как и прежде для исследования поведения этих операций я использовал язык FORTH и его реализацию WinForth32. Вот как выглядят приведенные выше логические операции на Форте с выводом промежуточных результатов

```
\ ------------------------------------------------------------------------
: .= ( x -- x ) ?PRINT IF DUP SPACE ." = " 8 HEX U.R THEN ;
\ ------------------------------------------------------------------------
: D-FF-BC ( c b -- x )
    ?PRINT IF 2DUP CR ." X & Y = " 8 HEX U.R SPACE ." & " 8 HEX U.R THEN
    AND .=
    ;
: D-FF-~B ( b -- ~b )
    ?PRINT IF DUP CR ." ~X = ~" 8 HEX U.R THEN
    INVERT .=
    ;
: D-FF-~BD ( ~b d -- x )
    ?PRINT IF 2DUP SWAP CR ." ~X & Z = " 8 HEX U.R SPACE ." & " 8 HEX U.R
THEN
    AND .=
    ;
: D-FF-F ( x1 x2 -- x3 )
    ?PRINT IF 2DUP SWAP CR ." . | .. = " 8 HEX U.R SPACE ." | " 8 HEX U.R
THEN
    OR .=
    ;
: 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-~D ( d -- ~d )
    ?PRINT IF DUP CR ." ~Z = ~" 8 HEX U.R THEN
    INVERT .=
    ;
: D-FG-~DC ( ~d c -- x )
    ?PRINT IF 2DUP CR ." Y & ~Z = " 8 U.R SPACE ." & " 8 U.R THEN
    AND .=
    ;
: D-FG-BD ( b d -- x )
    ?PRINT IF 2DUP CR ." X & Z = " SWAP 8 U.R SPACE ." & " 8 U.R THEN
    AND .=
    ;
: D-FG-BDv~DC ( bd ~dc -- x )
    ?PRINT IF 2DUP CR ." XZ | ~ZY = " SWAP 8 U.R SPACE ." | " 8 U.R THEN
    OR .=
    ;
: D-FG ( b-addr c-addr d-addr -- g )
    @ ROT @ ROT @ ROT ( b c d )
    DUP          ( b c d d )
    D-FG-~D      ( b c d xi )
    ROT          ( b d xi c )
    D-FG-~DC     ( b d xii )
    -ROT         ( xii b d )
    D-FG-BD      ( xii xiii )
    D-FG-BDv~DC  ( g )
    ;
\ ------------------------------------------------------------------------
: D-FH-B^C ( b c -- x )
    ?PRINT IF 2DUP CR ." X ^ Y = " SWAP 8 U.R SPACE ." ^ " 8 U.R THEN
    XOR .=
    ;
: D-FH-^D ( d x1 -- x2 )
    ?PRINT IF 2DUP CR ." . ^ Z = " 8 U.R SPACE ." ^ " 8 U.R THEN
    XOR .=
    ;
: D-FH ( b-addr c-addr d-addr -- h )
    @ ROT @ ROT @ ( d b c )
    D-FH-B^C ( d xi )
    D-FH-^D  ( h )
    ;
\ ------------------------------------------------------------------------
: D-FI-~D ( d -- ~d )
    ?PRINT IF DUP CR ." ~Z = ~" 8 HEX U.R THEN
    INVERT .=
    ;
: D-FI-Bv~D ( ~d b -- x )
    ?PRINT IF 2DUP CR ." X | ~Z = " 8 HEX U.R SPACE ." | " 8 HEX U.R THEN
    OR .=
    ;
: D-FI-^C ( c x1 -- x2 )
    ?PRINT IF 2DUP SWAP CR ." C ^ . = " 8 U.R SPACE ." ^ " 8 U.R THEN
    XOR .=
    ;
: D-FI ( b-addr c-addr d-addr -- i )
    @ ROT @ ROT @ ROT ( b c d )
    D-FI-~D    ( b c xi )
    ROT        ( c xi b )
    D-FI-Bv~D  ( c xii )
    D-FI-^C    ( i )
    ;
\ ------------------------------------------------------------------------
```

И оптимизированная версия

```
: 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 DUP INVERT ROT AND -ROT
AND OR ;
: FH ( b-addr c-addr d-addr -- h ) @ ROT @ ROT @ XOR XOR ;
: FI ( b-addr c-addr d-addr -- i ) @ ROT @ ROT @ ROT INVERT ROT OR XOR ;
```

В Форте очень удобно проверять как работает та или иная операция, например можно выполнить в интерактивном режиме такой запрос:

A B C D-FF

И в результате на экране будет выведена полная трассировка операции.

Если вы пожелаете вывести результат не в шестнадцатеричном, а в двоичном виде, то нужно везде заменить слово HEX на слово BINARY.

Преобразование блока данных

Для преобразования блока данных мы определим четыре базовых команды, которые будет выполнять MD5 процессор. Эти четыре базовые команды производят обработку и обновление регистров процессора на основании их предыдущего содержимого и на основании данных из БЛОЧНОГО БУФЕРА.

Эти операции описываются следующим выражением:

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

Здесь испольубтся следующие обозначения:
A, b, c, d – регистры MD5 процессора;
F – одна из определенных выше логических операций F, G, H или I;
X[k] – k-й регистр БЛОЧНОГО БУФЕРА;
T[I] – I-й элемент массива констант (о нем немного позже);
<<< -- операция циклического сдвига влево;
S – количество разрядов циклического сдвига.

На языке программирования Си в библиотеке OpenSSL это реализовано следующим образом:

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

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

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

```
#define R3(a,b,c,d,k,s,t) { \
        a+=((k)+(t)+I((b),(c),(d))); \
        a=ROTATE(a,s); \
        a+=b; };
```

Для изучения работы этих элементарных команд я использовал следующую реализацию на Форте:

```
: D-TRANSFORM-F+XK ( f k -- x )
    ?PRINT IF 2DUP CR ." F + X[" 2 DECIMAL .R ." ] = " 8 HEX U.R THEN
    CELLS  ( f ki )
    MD5_CTX_DATA
    +      ( f a-ki )
    @      ( f xi )
    ?PRINT IF DUP SPACE ." + " 8 HEX U.R THEN
    +      ( x )
    .=
    ;
: D-TRANSFORM-+TI ( i x1 -- x2 )
    ?PRINT IF 2DUP CR ." . + T[" SWAP 2 DECIMAL .R ." ] = " 8 HEX U.R THEN
    SWAP   ( x1 i )
    1-
    CELLS  ( x1 ii )
    MD5_T  ( x1 ii a-t )
    +      ( x1 a-ii )
    @      ( x1 xi )
    ?PRINT IF DUP SPACE ." + " 8 HEX U.R THEN
    +      ( x2 )
    .=
    ;
: D-TRANSFORM-+A ( a-addr x1 -- x2 )
    SWAP   ( x1 a-addr )
    @      ( x1 a )
    ?PRINT IF 2DUP CR ." A + . = " 8 HEX U.R SPACE ." + " 8 HEX U.R THEN
    + .=
    ;
: D-TRANSFORM-ROL ( s x1 -- x2 )
    ?PRINT IF 2DUP CR ." . <<< " OVER 2 DECIMAL .R SPACE ." = " 8 HEX U.R
SPACE ." <<< " 2 DECIMAL .R THEN
    SWAP
    LROTATE .=
    ;
: D-TRANSFORM-+B ( b-addr x1 -- x2 )
    SWAP   ( x1 b-addr )
    @      ( x1 b )
    ?PRINT IF 2DUP CR ." B + . = " 8 HEX U.R SPACE ." + " 8 HEX U.R THEN
    + .=
    ;
\ a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s)
: D-TRANSFORM ( b-addr s a-addr i f k -- ai )
    D-TRANSFORM-F+XK ( b-addr s a-addr i xi )  \ ( F(b,c,d) + X[k] )
    D-TRANSFORM-+TI  ( b-addr s a-addr xii )   \ ( + T[i] )
    D-TRANSFORM-+A   ( b-addr s xiii )         \ ( a + )
    D-TRANSFORM-ROL  ( b-addr xiv )            \ ( <<< s )
    D-TRANSFORM-+B   ( ai )                    \ ( b + )
    ;
```

или ее оптимизированную версию без отладочной печати

```
: TRANSFORM ( a-addr b-addr s a-addr i f k -- ai )
    CELLS MD5_CTX_DATA + @ + SWAP CELLS MD5_T + @ + SWAP @ +
```

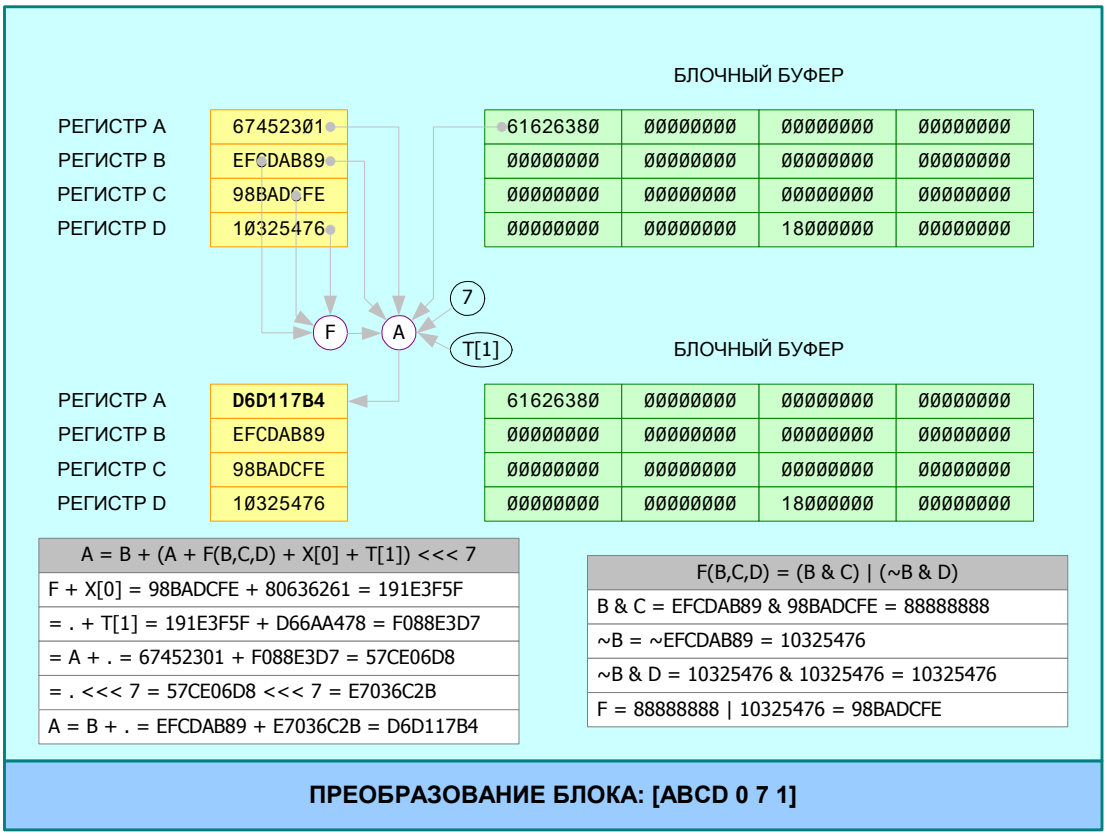
```
    SWAP LROTATE SWAP @ +
    ;
```

Для того, чтобы усилить алгоритм построения цифрового дайджеста элементарная операция преобразования данных одного регистра была изменена, по сравнению с алгоритмом MD4, а также в нее была добавлена таблица констант T из 64 элементов. Эта таблица добавляет некий "шум" к преобразуемым данным и представляет собой константы построенные на основании функции синус. Точнее, i-й элемент таблицы равен 4,294,967,296 * abs(sin(i)). Собственное в этом и состоит "сила" алгоритма MD5 по сравнению с алгоритмом MD4, однако за это усиление приходится расплачиваться большим количеством вычислений и большим объемом требуемой памяти.

Первой в преобразовании блока выполняется операция преобразования регистра A. Она выглядит следующим образом:

A = B + ((A + F(B,C,D) + X[0] + T[1]) <<< 7)

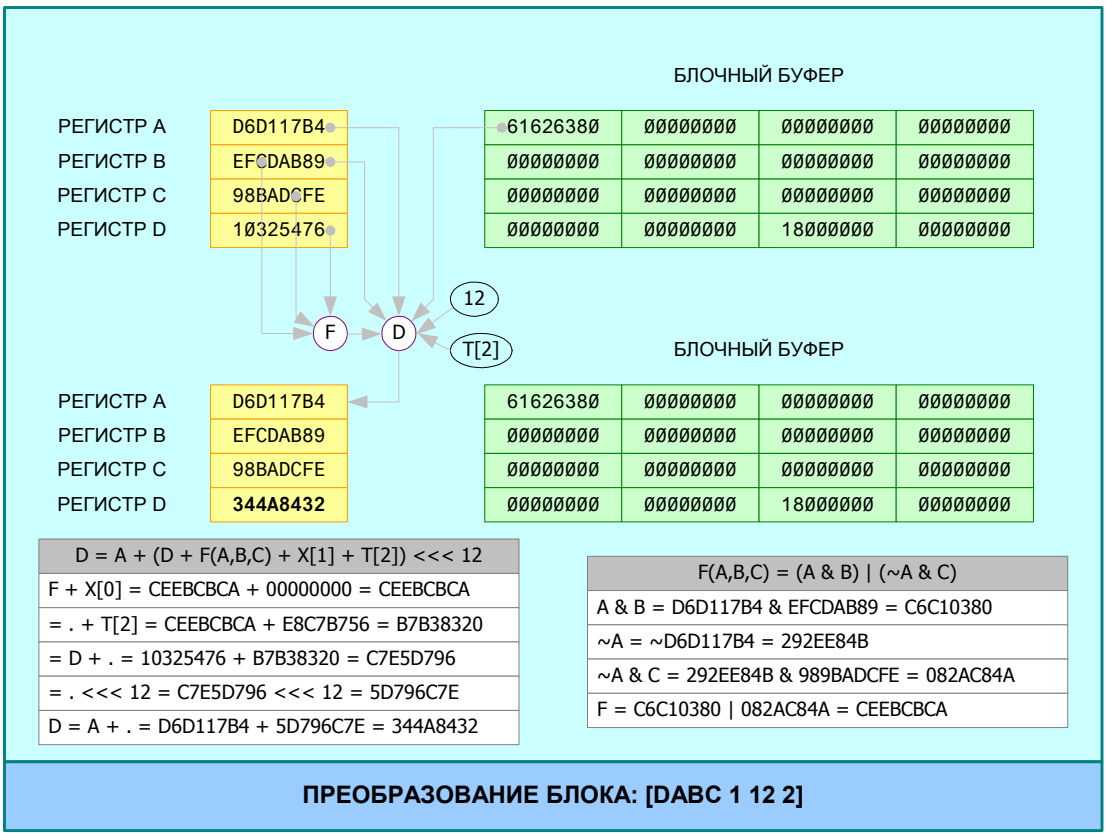
И проиллюстрирована на Рисунке 4.



Далее выполняется операция преобразования регистра D:

D = A + ((D + F(A,B,C) + X[1] + T[2]) <<< 12)

Которая проиллюстрирована на Рисунке 5.



Алгоритм диктует последовательность элементарных преобразований, которые организованы в четыре раунда. Каждый раунд использует свою логическую операцию, рассмотренную выше: Раунд 1 использует логическую операцию F, Раунд 2 использует логическую операцию G, Раунд 3 использует логическую операцию H, Раунд 4 использует логическую операцию I.

Пусть [abcd k s I] обозначает элементарную операцию преобразования. Тогда Раунд 1 представляет собой последовательность следующих операций:

```
[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]
```

Остальные три раунда приведены в Приложении.

В библиотеке OpenSSL этот раунд реализован следующим образом:

```
/* Round 0 */
R0(A,B,C,D,X[ 0], 7,0xd76aa478L);
R0(D,A,B,C,X[ 1],12,0xe8c7b756L);
R0(C,D,A,B,X[ 2],17,0x242070dbL);
R0(B,C,D,A,X[ 3],22,0xc1bdceeeL);
R0(A,B,C,D,X[ 4], 7,0xf57c0fafL);
R0(D,A,B,C,X[ 5],12,0x4787c62aL);
R0(C,D,A,B,X[ 6],17,0xa8304613L);
R0(B,C,D,A,X[ 7],22,0xfd469501L);
R0(A,B,C,D,X[ 8], 7,0x698098d8L);
R0(D,A,B,C,X[ 9],12,0x8b44f7afL);
R0(C,D,A,B,X[10],17,0xffff5bb1L);
R0(B,C,D,A,X[11],22,0x895cd7beL);
R0(A,B,C,D,X[12], 7,0x6b901122L);
```

```
      R0(D,A,B,C,X[13],12,0xfd987193L);
      R0(C,D,A,B,X[14],17,0xa679438eL);
      R0(B,C,D,A,X[15],22,0x49b40821L);
```

Для того же, чтобы отследить правильность работы процессора я опять-таки использовал код на языке Форт. Версия со выводом полной трассировочной информации выглядит так:

```
: D-STEPF1  ( -- ) RB  7 RA  1  RB RC RD D-FF  0 D-TRANSFORM RA !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF2  ( -- ) RA 12 RD  2  RA RB RC D-FF  1 D-TRANSFORM RD !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF3  ( -- ) RD 17 RC  3  RD RA RB D-FF  2 D-TRANSFORM RC !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF4  ( -- ) RC 22 RB  4  RC RD RA D-FF  3 D-TRANSFORM RB !  ?PRINT IF
.ABCD CR CR THEN ;

: D-STEPF5  ( -- ) RB  7 RA  5  RB RC RD D-FF  4 D-TRANSFORM RA !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF6  ( -- ) RA 12 RD  6  RA RB RC D-FF  5 D-TRANSFORM RD !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF7  ( -- ) RD 17 RC  7  RD RA RB D-FF  6 D-TRANSFORM RC !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF8  ( -- ) RC 22 RB  8  RC RD RA D-FF  7 D-TRANSFORM RB !  ?PRINT IF
.ABCD CR CR THEN ;

: D-STEPF9  ( -- ) RB  7 RA  9  RB RC RD D-FF  8 D-TRANSFORM RA !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF10 ( -- ) RA 12 RD 10  RA RB RC D-FF  9 D-TRANSFORM RD !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF11 ( -- ) RD 17 RC 11  RD RA RB D-FF 10 D-TRANSFORM RC !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF12 ( -- ) RC 22 RB 12  RC RD RA D-FF 11 D-TRANSFORM RB !  ?PRINT IF
.ABCD CR CR THEN ;

: D-STEPF13 ( -- ) RB  7 RA 13  RB RC RD D-FF 12 D-TRANSFORM RA !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF14 ( -- ) RA 12 RD 14  RA RB RC D-FF 13 D-TRANSFORM RD !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF15 ( -- ) RD 17 RC 15  RD RA RB D-FF 14 D-TRANSFORM RC !  ?PRINT IF
.ABCD CR CR THEN ;
: D-STEPF16 ( -- ) RC 22 RB 16  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
    ;
```

и оптимизированный вариант

```
: STEPF1  ( -- )  3 RA 0 RB RC RD FF  0 TRANSFORM RA ! ;
: STEPF2  ( -- )  7 RD 0 RA RB RC FF  1 TRANSFORM RD ! ;
: STEPF3  ( -- ) 11 RC 0 RD RA RB FF  2 TRANSFORM RC ! ;
: STEPF4  ( -- ) 19 RB 0 RC RD RA FF  3 TRANSFORM RB ! ;

: STEPF5  ( -- )  3 RA 0 RB RC RD FF  4 TRANSFORM RA ! ;
: STEPF6  ( -- )  7 RD 0 RA RB RC FF  5 TRANSFORM RD ! ;
: STEPF7  ( -- ) 11 RC 0 RD RA RB FF  6 TRANSFORM RC ! ;
```

```
: STEPF8  ( -- ) 19 RB 0 RC RD RA FF  7 TRANSFORM RB ! ;

: STEPF9  ( -- )  3 RA 0 RB RC RD FF  8 TRANSFORM RA ! ;
: STEPF10 ( -- )  7 RD 0 RA RB RC FF  9 TRANSFORM RD ! ;
: STEPF11 ( -- ) 11 RC 0 RD RA RB FF 10 TRANSFORM RC ! ;
: STEPF12 ( -- ) 19 RB 0 RC RD RA FF 11 TRANSFORM RB ! ;

: STEPF13 ( -- )  3 RA 0 RB RC RD FF 12 TRANSFORM RA ! ;
: STEPF14 ( -- )  7 RD 0 RA RB RC FF 13 TRANSFORM RD ! ;
: STEPF15 ( -- ) 11 RC 0 RD RA RB FF 14 TRANSFORM RC ! ;
: STEPF16 ( -- ) 19 RB 0 RC RD RA FF 15 TRANSFORM RB ! ;

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

Теперь можно в интерактивном режима инициализировать процессор (MD5-INIT) и выполнить только первый раунд (D-ROUND1). При этом на экране будет представлено содержимое всех регистров процессора во время работы.

Выполнение последовательно четырех раундов преобразований приводит у формированию промежуточного цифрового дайджеста сообщения в регистрах A, B, C и D.

Остальные раунды в статье не приведены. Их можно найти в Приложении, или в реализации библиотеки OpenSSL.

Как это реализовано в OpenSSL

Реализация примитивов в библиотеке OpenSSL использует тот же самый код, что и для алгоритма MD4. Исключение состоит лишь в том, что для построения промежуточного цифрового дайджеста блока данных их 16 байт используется другая функция, которая будет приведена ниже. Итак:

- 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 MD5_Init(MD5_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 MD5_Update(MD5_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 MD5_Final(unsigned char *md, MD5_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));
         */
        }
```

А функция преобразования одного 16-байтового блока данных, использующая рассмотренные выше операции преобразования регистров процессора выглядит так:

```
void md5_block_host_order(MD5_CTX *c, const void *data, int num)
        {
        const MD5_LONG *X=data;
        register unsigned long A,B,C,D;

        A=c->A;
        B=c->B;
        C=c->C;
        D=c->D;

        for (;num--;X+=HASH_LBLOCK)
                {

                /* Раунд 0 */
                R0(A,B,C,D,X[ 0], 7,0xd76aa478L);
                R0(D,A,B,C,X[ 1],12,0xe8c7b756L);
                R0(C,D,A,B,X[ 2],17,0x242070dbL);
                R0(B,C,D,A,X[ 3],22,0xc1bdceeeL);
                R0(A,B,C,D,X[ 4], 7,0xf57c0fafL);
                R0(D,A,B,C,X[ 5],12,0x4787c62aL);
                R0(C,D,A,B,X[ 6],17,0xa8304613L);
                R0(B,C,D,A,X[ 7],22,0xfd469501L);
                R0(A,B,C,D,X[ 8], 7,0x698098d8L);
                R0(D,A,B,C,X[ 9],12,0x8b44f7afL);
                R0(C,D,A,B,X[10],17,0xffff5bb1L);
                R0(B,C,D,A,X[11],22,0x895cd7beL);
                R0(A,B,C,D,X[12], 7,0x6b901122L);
                R0(D,A,B,C,X[13],12,0xfd987193L);
                R0(C,D,A,B,X[14],17,0xa679438eL);
                R0(B,C,D,A,X[15],22,0x49b40821L);

                /* Раунд 1 */
                R1(A,B,C,D,X[ 1], 5,0xf61e2562L);
                R1(D,A,B,C,X[ 6], 9,0xc040b340L);
                R1(C,D,A,B,X[11],14,0x265e5a51L);
                R1(B,C,D,A,X[ 0],20,0xe9b6c7aaL);
                R1(A,B,C,D,X[ 5], 5,0xd62f105dL);
                R1(D,A,B,C,X[10], 9,0x02441453L);
                R1(C,D,A,B,X[15],14,0xd8a1e681L);
                R1(B,C,D,A,X[ 4],20,0xe7d3fbc8L);
                R1(A,B,C,D,X[ 9], 5,0x21e1cde6L);
                R1(D,A,B,C,X[14], 9,0xc33707d6L);
                R1(C,D,A,B,X[ 3],14,0xf4d50d87L);
                R1(B,C,D,A,X[ 8],20,0x455a14edL);
                R1(A,B,C,D,X[13], 5,0xa9e3e905L);
                R1(D,A,B,C,X[ 2], 9,0xfcefa3f8L);
                R1(C,D,A,B,X[ 7],14,0x676f02d9L);
                R1(B,C,D,A,X[12],20,0x8d2a4c8aL);

                /* Раунд 2 */
                R2(A,B,C,D,X[ 5], 4,0xfffa3942L);
                R2(D,A,B,C,X[ 8],11,0x8771f681L);
                R2(C,D,A,B,X[11],16,0x6d9d6122L);
                R2(B,C,D,A,X[14],23,0xfde5380cL);
                R2(A,B,C,D,X[ 1], 4,0xa4beea44L);
```

```
            R2(D,A,B,C,X[ 4],11,0x4bdecfa9L);
            R2(C,D,A,B,X[ 7],16,0xf6bb4b60L);
            R2(B,C,D,A,X[10],23,0xbebfbc70L);
            R2(A,B,C,D,X[13], 4,0x289b7ec6L);
            R2(D,A,B,C,X[ 0],11,0xeaa127faL);
            R2(C,D,A,B,X[ 3],16,0xd4ef3085L);
            R2(B,C,D,A,X[ 6],23,0x04881d05L);
            R2(A,B,C,D,X[ 9], 4,0xd9d4d039L);
            R2(D,A,B,C,X[12],11,0xe6db99e5L);
            R2(C,D,A,B,X[15],16,0x1fa27cf8L);
            R2(B,C,D,A,X[ 2],23,0xc4ac5665L);

            /* Раунд 3 */
            R3(A,B,C,D,X[ 0], 6,0xf4292244L);
            R3(D,A,B,C,X[ 7],10,0x432aff97L);
            R3(C,D,A,B,X[14],15,0xab9423a7L);
            R3(B,C,D,A,X[ 5],21,0xfc93a039L);
            R3(A,B,C,D,X[12], 6,0x655b59c3L);
            R3(D,A,B,C,X[ 3],10,0x8f0ccc92L);
            R3(C,D,A,B,X[10],15,0xffeff47dL);
            R3(B,C,D,A,X[ 1],21,0x85845dd1L);
            R3(A,B,C,D,X[ 8], 6,0x6fa87e4fL);
            R3(D,A,B,C,X[15],10,0xfe2ce6e0L);
            R3(C,D,A,B,X[ 6],15,0xa3014314L);
            R3(B,C,D,A,X[13],21,0x4e0811a1L);
            R3(A,B,C,D,X[ 4], 6,0xf7537e82L);
            R3(D,A,B,C,X[11],10,0xbd3af235L);
            R3(C,D,A,B,X[ 2],15,0x2ad7d2bbL);
            R3(B,C,D,A,X[ 9],21,0xeb86d391L);

            A = c->A += A;
            B = c->B += B;
            C = c->C += C;
            D = c->D += D;

            }
      }
```

Именно здесь и сокрыто основное отличие в реализации двух алгоритмов.

Как это реализовано в Microsoft Windows

Microsoft Windows прячет от нас детали реализации алгоритма построения цифрового дайджеста MD5 и предлагает для работы только Crypto API – прикладной программный интерфейс для работы с криптографической библиотекой.

Crypto API построена по принципу так называемых "провайдеров", или в более русскоязычном варианте – модулей, обеспечивающих функции криптозащиты. Из этого следует, что вы можете построить свой собственный модуль, следуя требованиям библиотеки Crypto API, зарегистрировать его и использовать.

Но мы посмотрим только на ту часть библиотеки, которая позволит нам построить цифровой дайджест MD5.

Для построения цифрового дайджеста блока данных и для работы с полученным цифровым дайджестом Crypto API предлагает четыре метода:

- **CryptCreateHash** – функция используется для инициализации хэширования блока данных. Она возвращает дескриптор хэш-объекта, который затем можно использовать для хэширования данных.
- **CryptGetHashParam** – функция, используемая для извлечения значения хэша.
- **CryptHashData** – функция, выполняющая хэширование блока данных.

- **CryptDestroyHash** – функция для освобождения дескриптора полученного от функции создания хэш-объекта CryptCreatehash.

Метод CryptHashData используеься для построения промежуточного цифрового дайджеста блока дынных. Эта функция может быть вызвана несколько раз для обработки длинного сообщения, разбитого на блоки различной длины.

В качестве примера мы рассмотрим построение цифрового дайджеста для блока данных длиной dwBufferLen, которые расположены в памяти по адресу pBuffer. Предполагается, что данные подлежащие хэшированию полностью расположены в буфере. Поэтому после извлечения цифрового дайджеста с помощью метода CryptoGetHashParam дальнейшее хэширование данных с помощью данного хэш-объекта более не возможно.

```
#include <wincrypt.h>      // определения CryptoAPI
/*
Значение некоторых констант, используемых в данном примере:
#define ALG_CLASS_HASH                  (4 << 13)
#define ALG_TYPE_ANY                    (0)
#define ALG_SID_MD5                     3
#define CALG_MD5        (ALG_CLASS_HASH | ALG_TYPE_ANY | ALG_SID_MD5)
#define HP_HASHVAL              0x0002  // Значение цифрового дайджеста
#define HP_HASHSIZE             0x0004  // Длина цифрового дайджеста
*/
BOOL bResult;
HCRYPTHASH hHash;
DWORD dwBufferLen;
DWORD dwValue;
PBYTE pBuffer;

// Получить дескриптор к хэш-объекту.
bResult = CryptCreateHash(
            hProv,                  // Дескриптор к CSP полученный ранее
            CALG_MD5,               // Алгоритм хэширования
            0,
            0,                      // Должен быть нулем!
            &hHash);                // Переменная для хранения дескриптора

// Построение цифрового дайджеста для блока данных.
bResult = CryptHashData(
            hHash,                  // Дескриптор к хэш-объекту
            pBuffer,                // Указатель на буфер данных
            dwBufferLen,            // Размер блока данных
            0);                     // Никаких специальных флагов

// Получить размер (длину) цифрового дайджеста.
dwBufferSize = sizeof(DWORD);
bResult = CryptGetHashParam(
            hHash,                  // Дескриптор к хэш-объекту
            HP_HASHSIZE,            // Получить размер дайджеста
            &dwValue,               // Буфер для хранения размера дайджеста
            &dwBufferLen,           // размер буфера данных
            0);                     // Должен быть нулем!

// Создать буфер для хранения цифрового дайджеста.
pBuffer = new char [dwBufferSize];

// Get hash value.
bResult = CryptGetHashParam(
            hHash,                  // Дескриптор к хэш-объекту
            HP_HASHVAL,             // Получить цифровой дайджест
            pBuffer,                // Буфер для хранения цифрового дайджеста
            &dwBufferSize,          // Размер данных
            0);                     // Должен быть нулем!
```

```
// Освободить хэш-объект.
CryptDestroyHash(hHash);
```

Вот и все! Несколько сложнее, немного более сложные вызовы, но результат одинаков – цифровой дайджест MD5 для блока данных. Стоит также отметить, что за этой сложностью вызовов и количеством параметров стоит гибкость CryptoAPI – всего четыре метода для построения цифровых дайджестов, будь то MD4, MD5, SHA-1 или ваш собственный, так как «провайдер» позволяет установить ваш собственный алгоритм хэширования.

Построение цифрового дайджеста строки

Код для построения цифрового дайджеста строки полностью аналогичен коду построения цифрового дайджеста строки для алгоритма MD4, за исключением того, что в именах вызовов функций MD4_* заменено на MD5_*:

```
MD5_CTX context;
unsigned char digest[16];
unsigned char string = "abc";
unsigned int len = strlen (string);

MD5_Init(&context);
MD5_Update(&context, string, len);
MD5_Final(digest, &context);
```

Ну и конечно же пример:

MD5("abc") = 900150983CD24FB0D6963F7D28E17F72

Построение цифрового дайджеста файла

Цифровой дайджест MD5 файла строится аналогично тому, как мы строили цифровой дайджест MD4:

```
FILE *file;
MD5_CTX context;
int len;
unsigned char buffer[1024], digest[16];

file = fopen(filename, "rb");

MD5_Init(&context);
while (len = fread (buffer, 1, 1024, file))
    {
    MD5_Update(&context, buffer, len);
    }
MD5_Final(digest, &context);

fclose(file);
```

Отличия состоят, как вы видите, только в наименованиях функций, а еще точнее только в одном символе: символ "4" заменен на "5".

Скорость работы и производительность

Здесь я не буду приводить графики аналогичные графикам для алгоритма построений цифрового дайджеста MD4. Я приведу две таблицы, показывающие на сколько алгоритм MD5 сложнее, чем алгоритм MD4. Эти таблицы показывают время, затраченное на построение 10,000 раз цифровых дайджестов MD4 и MD5 сообщения размером 10,000 байт, и, соответственно, пропускную способность алгоритма. Приведены два варианта: один для разбивки сообщения на куски размером 64 байта, а другой – на 128 байт. (Помните? Размер буфера нужно брать кратным 64!) Вычисления проводились на компьютере с процессором Intel Pentium III 750 МГц, работающем под управлением операционной системы Microsoft Windows XP.

MD4				
	64		128	
RFC	2.574 сек	37940 кБ/сек	2.234 сек	43717 кБ/сек
OpenSSL	0.891 сек	109603 кБ/сек	0.781 сек	125040 кБ/сек

MD5				
	64		128	
RFC	2.614 сек	37359 кБ/сек	2.344 сек	41662 кБ/сек
OpenSSL	1.152 сек	84771 кБ/сек	1.092 сек	89429 кБ/сек

Анализ этих двух таблиц показывает, что:

- Алгоритм MD5 медленнее алгоритма MD4 в реализации RFC на 1.55%..4.92%
- И, соответственно, алгоритм MD5 менее производителен, чем алгоритм MD4 на 1.53%..4.69%
- Алгоритм MD5 медленнее алгоритма MD4 в реализации RFC на 29.29%..39.82%
- Алгоритм MD5 менее производителен, чем алгоритм MD4 на 22.66%..28.48%.

Как видите, для более быстрой реализации (OpenSSL), цифры снижения производительности более значительны. Можно с уверенностью сказать, что для быстрой реализации алгоритм MD5 на 30% медленнее, чем алгоритм MD4!

Свойства односторонней хэш функции

Одностороння хэш функция обладает двумя очень важными свойствами.

Во-первых, такая функция является трудно-коллизируемой. Сложное получилось слово, поэтому оно требует дополнительного пояснения: очень сложно вычислительными методами найти два различных сообщения, которые дадут одинаковый цифровой дайджест.

И, во-вторых, имея цифровой дайджест сообщения невозможно восстановить исходное сообщение из которого он был построен.

Рассмотрим теперь каждое из этих свойств более подробно.

Невозможность восстановления исходного сообщения

Чтобы пояснить невозможность такой операции ограничимся рассмотрением исходных сообщений длиной в 256 бит, что всего-лишь в 2 раза больше, чем размер цифрового дайджеста MD5.

Итак, возможно 2^{256} различных сообщений. В то же время, возможно всего 2^{128} различных вариантов цифрового дайджеста MD5. Изобразим это в виде двух множеств M (от Messages) и D (от Digests) (смотрите Рисунок 5).

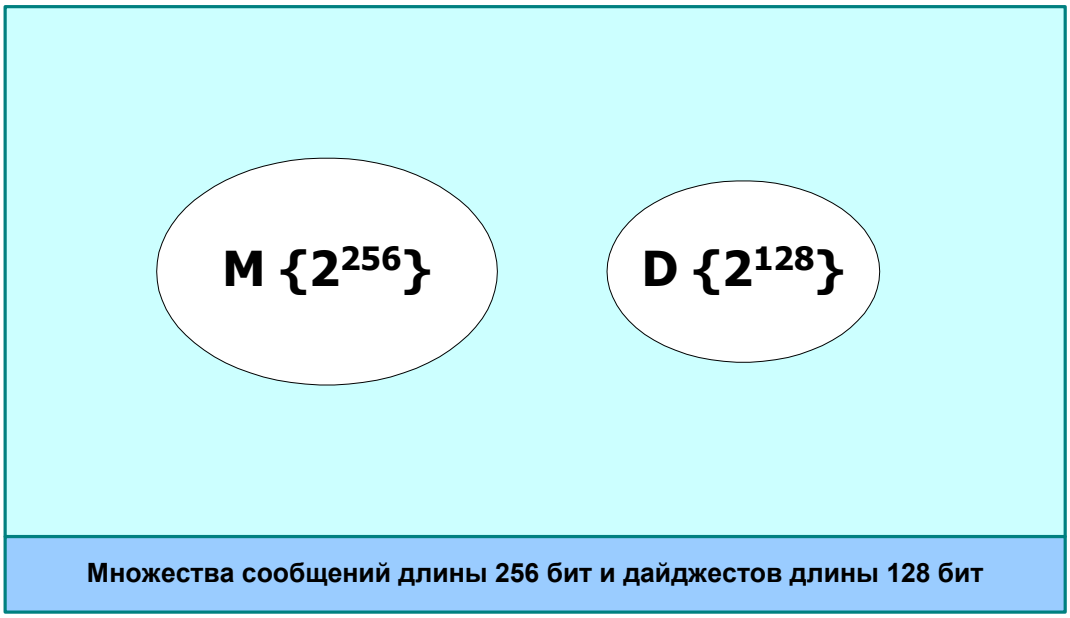


Рисунок 5. Множество всех сообщений длиной 256 бит и множество всех дайджестов длиной 128 бит.

Так как множество M больше, чем множество D, а мы строим однозначне соответствие между элементами множества M и множества D, то на 2^{128}-м элементе множества M множество D исчерпается. Поэтому нам придется назначать уже назначенные ранее элементы множества D. В результате, каким-либо элементам множества D будет соответствовать более одного элемента множества M (смотрите Рисунок 6).

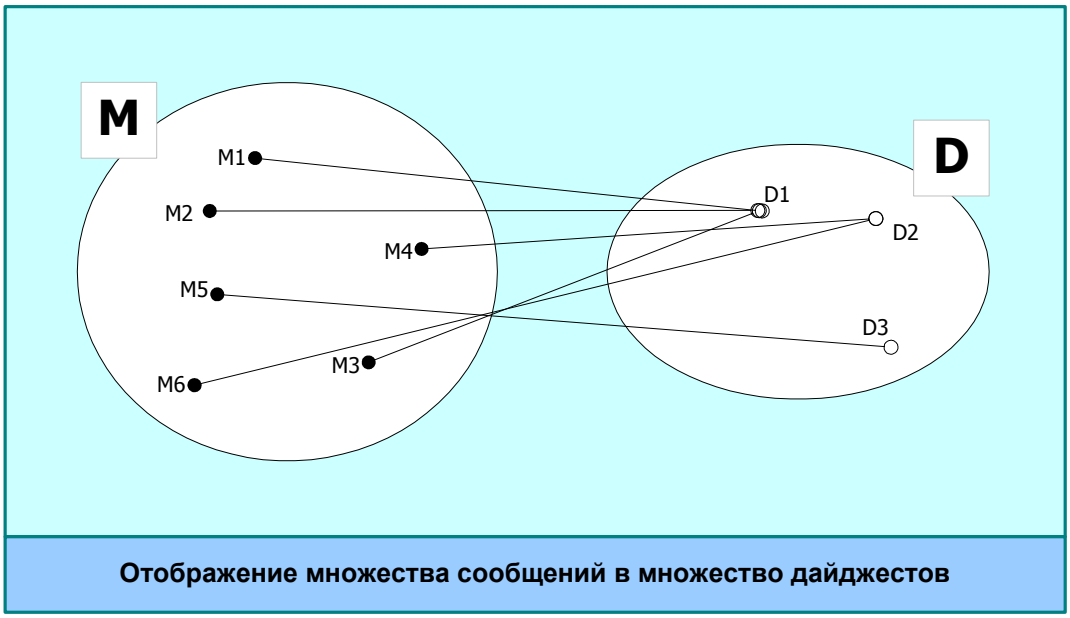


Рисунок 6. Множества M и D после отображения элементов одного в другое.

Теперь, если мы попытаемся определить, какой элемент множества M соответствует элементу D1 множества D, то мы увидим, что ему может соответствовать один из трех элементов множества M: M1, M2 или M3, то есть обратного однозначного соответствия нет и это подтверждает то, что невозможно восстановить исходное сообщение по его цифровому дайджесту.

Коллизии

Теперь о втором свойстве хэш функции. Как мы только что видели коллизии просто неизбежны, но очень сложно найти правила по которым можно построить два сообщения, которые будут иметь одинаковый цифровой дайджест.

Конечно, здесь многие могут воскликнуть: "Так давайте переберем все 2^{256} сообщений и посмотрим на их дайджесты! Компьютер-то у нас есть! Пускай он это и сделает!"

И вот здесь мы переходим в область так называемых больших чисел.....

Давайте для начала вспомним с какой скоростью работал наш тест для алгоритма MD4 – 125040 килобайт в секунду, на компьютере с процессором Intel Pentium III работающем на частоте 750 МГц.

А теперь будем считать:
- $2^{256} \approx 1.1579 \cdot 10^{77}$
- размер сообщения 256 бит, то есть 16 байт, тогда преобразование будет выполняться в результате выполнения одного блока размером 64 байта
- тогда нам нужно будет преобразовать $1.1579 \cdot 10^{77} \times 4 = 4.613 \cdot 10^{77}$ байт (блок в 4 раза больше, чем размер сообщения) информации
- $4.613 \cdot 10^{77}$ байт ÷ 1024 = $4.5230 \cdot 10^{74}$ килобайт
- делим полученный объем преобразования требуемых данных на скорость преобразования, имеем $4.5230 \cdot 10^{74}$ килобайт ÷ 125040 килобайт в секунду = $3.6173 \cdot 10^{69}$ секунд ☺
- что составляет ☺ $3.6173 \cdot 10^{69}$ секунд ÷ количество секунд в году = $3.6173 \cdot 10^{69}$ секунд ÷ (60 [секунд в минуте] × 60 [минут в часе] × 24 [часов в сутках] × 365 [дней в году]) = $3.6173 \cdot 10^{69}$ секунд ÷ 31,536,000 секунд в году = **$1.1470 \cdot 10^{62}$ лет** ☺

Ну так как? Начнем?

А теперь посмотрим сколько нам нужно памяти, чтобы сохранить все полученные дайджесты (нам ведь нужно будет сравнить их между собой, чтобы найти одинаковые):
- размер MD5 дайджеста равен 128 бит = 16 байт
- нам нужно сохранить 2^{256} дайджестов
- итого 2^{256} * 16 байт = 2^{260} байт
- 2^{260} байт ÷ 1024 = 2^{250} килобайт
- 2^{250} килобайт ÷ 1024 = 2^{240} мегабайт
- 2^{240} мегабайт ÷ 1024 = 2^{230} гигабайт

Тоже немало! ☺

То есть, как видно из простых расчетов, вычислительными средствами существующими сегодня практически невозможно подобрать сообщение по его дайджесту. Однако все же должны существовать правила по которым можно подобрать два сообщения с одинаковым дайджестом (ибо алгоритм полностью детерминирован) – в этом и состоит задача криптоанализа.

Эта задача непроста так как изменение одного бита в исходном сообщении приводит к изменению большого количества бит в дайджесте. Например,

MD5(abc) = MD5(616263_{16}) = MD5($101000011010001010100011_2$) =
= **900150983CD24FB0D6963F7D28E17F72**
MD5(acc) = MD5(616363_{16}) = MD5($101000011010001**1**10100011_2$) =
= **1673448EE7064C989D02579C534F6B66**

10010000000000010101000010011000011...
00010**1**1**0**0**1**1**1**00**1**1010**0**0**1**00100**0**1**1**1**0**1**1**1**0**...

Жирным шрифтом выделены изменившиеся разряды.

Однако об одном из возможных взломов дайджеста MD5 стоит сказать.

Считается ли алгоритм MD5 взломанным?

В 1996 году появилось сообщение о криптоанализе алгоритма MD5. основная предпосылка выглядит следующим образом: если бы была возможность задать отличные начальные значения регистров MD5 процессора от тех, которые заложены в алгоритме (помните, 0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476), то затем можно было бы подобрать два различных сообщения, которые различаются только может быть в нескольких разрядах, таких, для которых может быть построен один и тот же дайджест. Что, собственно, и было сделано в этом криптоанализе. Математически это выражается так:

MD5(IV, M1) = MD5(IV, M2),

Где IV = Initial Values = начальные значения, M1 и M2 различные сообщения.

Автор сообщения, Ганс Доббертин (Hans Dobbertin), нашел, что если использовать следующие начальные значения регистров MD5 процессора

IV_A = 0x12AC2375
IV_B = 0x3B341042
IV_C = 0x5F62B97C
IV_D = 0x4BA763ED

И задать значение блока данных для преобразования следующим образом:

X0 = 0xAA1DDABE
X1 = 0xD97ABFF5
X2 = 0xBBF0E1C1
X3 = 0x32774244
X4 = 0x1006363E
X5 = 0x7218209D
X6 = 0xE01C136D
X7 = 0x9DA64D0E
X8 = 0x98A1FB19
X9 = 0x1FAE44B0
X10 = 0x236BB992
X11 = 0x6B7A779B
X12 = 0x1326ED65
X13 = 0xD93E0972
X14 = 0xD458C868
X15 = 0x6B72746A

То второе сообщение можно построить из первого с помощью формулы:

$$X'_i = \begin{cases} X_i, & i < 16, i \neq 14; \\ X_{14} + 2^9, & i = 14. \end{cases}$$

Тогда MD5(IV, X) = MD5(IV, X') = BF90E670752AF92B9CE4E3E1B12CF8DE.

С моей точки зрения этот криптоанализ имеет только теоретическое значение, так как алгоритмом и RFC 1321 диктуется, что начальные значение регистров MD5 процессора должны иметь определенные значения и во всех реализациях они будут одинаковыми.

Но тем не менее, некоторые авторы считают алгоритм MD5 взломанным!

К сожалению, я не проверял приведенные выше данные и вычисления. Если у кого возникнет такое желание, то он может воспользоваться набором слов Форта, которые я использовал для анализа работы алгоритма. Тексты слов Форта для процессоров MD4 и MD5 можно получить с сайта журнала HTTP://WWW.Relcom.COM/Argc-Argv.

Резюме

Редакция 2 от 7 февраля 2003 года
Ло Шу (LoShu_LoShu@Yahoo.CO.UK, LoShu_LoShu@HotMail.COM).

Приложение. RFC 1321

Network Working Group
Request for Comments: 1321

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

Алгоритм построения цифрового дайджеста сообщения MD5

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), Дэвиду Чауму (David Chaum) и Ноаму Нисану (Noam Nisan) за их многочисленные важные комментарии, замечания и предложения.

Содержание

1. Резюме руководства

Этот документ описывает алгоритм построения цифрового дайджеста MD5. 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 MD5 предназначается для приложений требующих использования цифровых подписей, в которых большие файлы должны быть "сжаты" безопасным способом перед тем как они будут зашифрованы закрытым (секретным) ключом с использованием таких криптосистем с открытым ключом, как, например, RSA.

Алгоритм MD5 спроектирован таким образом, чтобы быть достаточно быстрым на 32-разрядных машинах. В дополнение к этому, алгоритм MD5 не требует никаких больших таблиц подстановки; алгоритм может быть достаточно компактно закодирован.

Алгоритм MD5 это дальнейшее развитие и расширение алгоритма MD4 [1,2]. MD5 работает несколько медленнее, чем MD4, но он и более "консервативен" в своем дизайне. Так как MD4 был спроектирован с целью быть исключительно быстрым алгоритмом, он был, что называется "на грани" в терминах риска от успешных криптоаналитических атак. MD5 наоборот, проигрывая немного в скорости дает большой выигрыш в безопасности. Он реализует некоторые из предложений сделанные многими специалистами, и содержит дополнительную оптимизацию. Алгоритм MD4 предложен для обзора общественности и возможной адаптации его в качестве стандарта.

Для приложений поддерживающих стандарт OSI, идентификатор объекта MD5 имеет вид

```
   md5 OBJECT IDENTIFIER ::=
     iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2)
5}
```

В типе AlgorithmIdentifier [3] из X.509, параметры для MD5 должны иметь тип NULL.

2. Терминология и нотация

В этом документе "слово" обозначает 32-разрядную величину, 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 MD5

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 четыре 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) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(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. (???) 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.

Функции G, H и I аналогичны функции F в том, что они воздействуют "параллельно" на биты чтобы построить результирующее значение на основании битов из X, Y и Z, таким образом, что если соответствующие биты X, Y и Z независимы, то и каждый бит в G(X,Y,Z), H(X,Y,Z) и I(X,Y,Z) также будет независим. Заметьте, что функция H является функцией исключающего ИЛИ или "четности" над своими аргументами.

Этот шаг использует таблицу из 64 элементов T[1 ... 64], построенную из функции синус. Пусть T[i] обозначает i-й элемент таблицы, который равен целой части от 4294967296 умноженного на abs(sin(i)), где I измеряется в радианах. Элементы таблицы приведены в приложении.

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 i] обозначает операцию
          a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
     /* Выполнить следующие 16 операций. */
     [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]

     /* Раунд 2. */
     /* Пусть [abcd k s i] обозначает операцию
          a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
     /* Выполнить следующие 16 операций. */
     [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]

     /* Раунд 3. */
     /* Пусть [abcd k s t] обозначает операцию
          a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
     /* Выполнить следующие 16 операций. */
     [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]

     /* Раунд 4. */
     /* Пусть [abcd k s t] обозначает операцию
          a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
     /* Выполнить следующие 16 операций. */
     [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]

     /* 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 */
```

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 MD5. Пример реализации на языке программирования Си приведен в приложении.

4. Резюме

Алгоритм построения цифрового дайджеста сообщения MD5 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 MD5 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.

5. Различия между MD4 и MD5

Следующий список представляет собой различия между MD4 и MD5:

1. Был добавлен четвертый раунд обработки.

2. Теперь каждый шаг содержит уникальную добавочную константу.

3. Функция g в раунде 2 была изменена с (XY v XZ v YZ) на (XZ v Y not(Z)), чтобы сделать g менее симметричной.

4. Теперь на каждом шаге добавляется результат из предыдущего шага. Это обеспечивает более быстрый "эффект обвала". (This promotes a faster "avalanche effect".)

5. Изменен порядок в котором обрабатываются слова в раундах 2 и 3, для того чтобы сделать их менее похожими друг на друга.

6. Количество сдвигов в каждом раунде было приблизительно оптимизировано, чтобы ускорить "эффект обвала". В различных раундах эти сдвиги различны.

Литература

[1] Rivest, R., "The MD4 Message Digest Algorithm", RFC 1320, MIT and

RSA Data Security, Inc., April 1992.

[2] 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.

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

ПРИЛОЖЕНИЕ A – Пример реализации

Это приложение содержит следующие файлы, которые были взяты из RSAREF: A Cryptographic Toolkit for Privacy-Enhanced Mail:

global.h – глобальный заголовочный файл

md5.h – заголовочный файл для MD5

md5c.c – исходный код для MD5

Чтобы получить более детальную информацию о RSAREF отправьте сообщение по адресу <rsaref@rsa.com>.

Приложение также содержит следующие файлы:

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 md5.h

```
/* MD5.H - заголовочный файл для MD5C.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. MD5 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. MD5 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.
 */

/* Контекст MD5. */
typedef struct {
  UINT4 state[4];                                   /* состояние (ABCD)
*/
  UINT4 count[2];      /* количество бит по модулю 2^64 (lsb first)
*/
  unsigned char buffer[64];                         /* буфер ввода
*/
} MD5_CTX;

void MD5Init PROTO_LIST ((MD5_CTX *));
void MD5Update PROTO_LIST
  ((MD5_CTX *, unsigned char *, unsigned int));
```

```
void MD5Final PROTO_LIST ((unsigned char [16], MD5_CTX *));
```

A.3 md5c.c

```
/* MD5C.C - RSA Data Security, Inc., Алгоритм построения цифрового
дайджеста
   сообщения MD5.
*/

/* 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. MD5 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. MD5 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 "md5.h"

/* Константы для процедуры MD5Transform.
 */

#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

static void MD5Transform 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 MD5_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD5_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
};

/* F, G, H и I - основные функции MD5.
 */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

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

/* FF, GG, HH, и II это преобразования для раундов 1, 2, 3, и 4. *\
/* Вращение реализовано отдельно от сложения, чтобы предотвратить
 повторное вычисление
 */
#define FF(a, b, c, d, x, s, ac) { \
 (a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
(a) += (b); \
  }
#define GG(a, b, c, d, x, s, ac) { \
 (a) += G ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define HH(a, b, c, d, x, s, ac) { \
 (a) += H ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define II(a, b, c, d, x, s, ac) { \
 (a) += I ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }

/* Инициализация MD5. Начинает операцию MD5, создавая новый контекст.
*/
void MD5Init (context)
MD5_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 MD5 обновления блока. Продолжает операцию построения
цифрового
 дайджеста сообщения MD5, обрабатывая очередной блок сообщения,
 и обновляя контекст.
*/
void MD5Update (context, input, inputLen)
MD5_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) {
 MD5_memcpy
   ((POINTER)&context->buffer[index], (POINTER)input, partLen);
 MD5Transform (context->state, context->buffer);

 for (i = partLen; i + 63 < inputLen; i += 64)
   MD5Transform (context->state, &input[i]);

 index = 0;
  }
  else
 i = 0;

  /* Поместить остаток входных данных в буфер */
  MD5_memcpy
 ((POINTER)&context->buffer[index], (POINTER)&input[i],
  inputLen-i);
}

/* Окончание работы MD5. Завершает операцию построения цифрового
дайджеста
сообщения MD5, записывая дайджест сообщения и обнуляя контекст.
*/
void MD5Final (digest, context)
unsigned char digest[16];                        /* дайджест сообщения */
MD5_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);
  MD5Update (context, PADDING, padLen);

  /* Добавить длину сообщения (до расширения) */
  MD5Update (context, bits, 8);

  /* Сохранить состояние в дайджест */
  Encode (digest, context->state, 16);

  /* Обнулить контекст.
*/
  MD5_memset ((POINTER)context, 0, sizeof (*context));
}

/* Основное преобразование MD5. Преобразует state основываясь на
block.
*/
static void MD5Transform (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, 0xd76aa478); /* 1 */
  FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
  FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
  FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
  FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
  FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
  FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
  FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
  FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
  FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
  FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
  FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
  FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
  FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
  FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
  FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

 /* Раунд 2 */
  GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
  GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
  GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
  GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
  GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
  GG (d, a, b, c, x[10], S22,  0x2441453); /* 22 */
  GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
  GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
  GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
  GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
  GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
```

```
  GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
  GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
  GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
  GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
  GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

  /* Раунд 3 */
  HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
  HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
  HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
  HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
  HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
  HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
  HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
  HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
  HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
  HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
  HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
  HH (b, c, d, a, x[ 6], S34,  0x4881d05); /* 44 */
  HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
  HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
  HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
  HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

  /* Раунд 4 */
  II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
  II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
  II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
  II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
  II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
  II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
  II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
  II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
  II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
  II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
  II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
  II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
  II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
  II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
  II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
  II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

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

  /* Заполнить нулями область с важной информацией.
*/
  MD5_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 MD5_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 MD5_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
 ("1234567890123456789012345678901234567890\
12345678901234567890123456789012345678890");
}

/* 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 MD5 (опция тестовой программы "-x") должен печатать следующие результаты:

```
MD5 test suite:
MD5 ("") = d41d8cd98f00b204e9800998ecf8427e
MD5 ("a") = 0cc175b9c0f1b6a831c399e269772661
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5
("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
d174ab98d277d9f5a5611c2c9f419d9f
```

```
MD5
("12345678901234567890123456789012345678901234567890123456789012345678
78901234567890") = 57edf4a22be3c955ac49da2e2107b67a
```

Соглашения по безопасности

Уровень безопасности обсуждаемый в данном документе предполагается достаточным для реализации гибридной схемы цифровой подписи очень высокой степени защиты основанной на MD5 и криптосистеме с открытым ключом.

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