A Method for Pre-Processing

Message Digest Input

FORTNER, James

Abstract

This paper describes a method for improving the reliability of standard message digest algorithms by pre-processing the input message. The message digest is then computed over the output of the pre-processing routine, thereby increasing the difficulty of forcing two messages to share a digest value.

Objectives

The key objective of this method was to reduce the ability to artificially product two documents that have the same message digest, thus making it more difficult to manipulate message content for whatever reason. The goal was to provide enhanced security using a method that would not consume excessive amounts of processor time and that could be easily implemented in hardware or firmware. As it turns out, this method can actually reduce the amount of overall processing required to compute the digest of larger messages.

The Problem

Most standard message digests process data in fixed-length blocks. A message is broken into these blocks and an initial value is assigned to the “partial result” of the digest. The result of processing each input message block is a function of the contents of that block and the partial result at the start of the processing for that block. The partial result after processing each block of the message is used as input when processing the next block of the message, with the final result being the digest value for the entire message.

Given that the input partial result is the same, differences in the input message blocks will result in different partial values after the blocks have been processed. Similarly, if the input partial results are different, the identical message blocks will result in different partial results at the end of processing. Thus, a single change in a message stream will result in different partial results starting with the message block containing the change and all subsequent blocks will yield different partial results due to the change.

If, however, a change in the input message were to be followed by another carefully crafted change that causes subsequent partial results to converge, the change would be hidden with respect to the final message digest value. Such a method for masking changes in a message stream and compensating for the change later in the message has already been demonstrated.

This vulnerability is a result of the block-at-a-time method of directly processing the raw input message stream.

The Proposal

This proposed solution involves using the input bytes of the message to direct the manipulation of elements of an array. The array will have two index values, one that is incremented by the value of the corresponding array element and one that is incremented by the value of the input message byte. After each index is incremented, the corresponding values are exchanged and the process is repeated. After the final message byte has been processed, the bit-length of the message, the two index values, and the permutated array are passed to the standard message digest routine.

The resulting digest value, although not computed directly over the message itself, will be a consistent value for any given message. Furthermore, the process is resistant to re-convergence to compensate for a change due to the index values and overall length being computationally part of the final digest.

The preprocessing is also simple enough to allow hardware or microcode implementation and, since the computationally intensive digest function is performed over a fixed-size set of data, it requires less resources to process large messages.

Implementation

The original concept was to compute a message digest over the resulting tables after processing. The message digest computation would be faster for large files since the input would be reduced to the 264 (or 520) bytes of pre-processing output. The output of pre-processing can also be appended to the message text.

Code Sample

The pre-processing method is implemented in the following code sample:

```
#ifdef   __PreDigest_Build
#define  PREDIGESTDLL
__declspec(dllexport)
#else
#define  PREDIGESTDLL
__declspec(dllimport)
#endif

 PREDIGESTDLL size_t  PMDSIZE( void
);
 PREDIGESTDLL int     PMDINIT( char
*scratch, long  bytelen );
 PREDIGESTDLL int     PMDDATA( char
*scratch, long  bytelen, char *
message );
 PREDIGESTDLL char   *PMDDONE( char
*scratch );#define   CTL         256

 PREDIGESTDLL
 size_t   PMDSIZE( void )  {
    return sizeof( PREDIGEST );
    }

 PREDIGESTDLL
 int      PMDINIT( char *scratch,
long  bytelen )  {
    PREDIGEST *w;
    int        loop;
    unsigned char   *scan;
    if ( bytelen < sizeof( PREDIGEST
) )
       return sizeof( PREDIGEST );
    w = (PREDIGEST *)scratch;
    w->bytelen =   0;
    w->aix     =   0;
    w->dix     = 128;
    memcpy( w->array, seed, CTL );
    return 0;
    }

 PREDIGESTDLL
 int      PMDDATA( char *scratch,
long  bytelen, char *message )  {

    PREDIGEST *w;
    unsigned char t;
    if ( bytelen < 0 )
       return -1;
    w = (PREDIGEST *)scratch;
    w->bytelen += bytelen;
    while ( bytelen-- )  {
       w->aix = (CTL-1) & ( w->aix +
w->array[w->aix] );
       w->dix = (CTL-1) & ( w->dix +
(unsigned char)*message);
       t                 = w-
>array[w->aix];
       w->array[w->aix] = w-
>array[w->dix];
       w->array[w->dix] = t;
       message++;
       }
    return 0;
    }

 PREDIGESTDLL
 char    *PMDDONE( char *scratch )
{
    return scratch;
    }
```

Example 1

The following example shows the work area before and after a short message. The array size has been set to 256 and the initialization data has been changed to help identify changes.

```
00000000    0000    0080

00 01 02 03 04 05 06 07 08
09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18
19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28
29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38
39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48
49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58
59 5A 5B 5C 5D 5E 5F
60 61 62 63 64 65 66 67 68
69 6A 6B 6C 6D 6E 6F
70 71 72 73 74 75 76 77 78
79 7A 7B 7C 7D 7E 7F
80 81 82 83 84 85 86 87 88
89 8A 8B 8C 8D 8E 8F
90 91 92 93 94 95 96 97 98
99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8
A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8
B9 BA BB BC BD BE BF
C0 C1 C2 C3 C4 C5 C6 C7 C8
C9 CA CB CC CD CE CF
D0 D1 D2 D3 D4 D5 D6 D7 D8
D9 DA DB DC DD DE DF
E0 E1 E2 E3 E4 E5 E6 E7 E8
E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8
F9 FA FB FC FD FE FF
```

This is a test message.

```
00000017    0083    00A8

D4 52 B6 03 04 A1 06 07 08
7A 0A 0B 0C 0D 0E 0F
A5 11 12 13 A6 F4 16 17 B5
19 1A 1B 1C 1D 1E DA
20 21 22 23 24 25 26 27 28
9B 2A 2B 2C 2D 2E 2F
30 31 32 33 BA 35 36 37 CD
39 3A 3B 00 3D 3E 3F
40 41 42 43 44 45 AE 47 48
49 4A 4B 4C F9 4E 4F
50 51 01 75 54 55 56 57 58
59 5A 5B 5C 5D 5E 5F
60 61 62 63 64 65 66 B8 68
69 6A 6B 6C 6D 6E 6F
70 71 72 73 74 53 76 77 78
79 09 7B 7C 7D 7E 7F
80 81 82 A8 84 85 86 87 88
89 8A 8B 8C 8D C4 8F
90 91 92 93 94 C8 96 97 98
99 9A 29 9C 9D 9E 9F
A0 05 A2 A3 A4 10 14 A7 18
A9 AA AB AC AD 46 AF
B0 B1 B2 B3 B4 83 02 B7 67
B9 34 BB BC BD BE BF
C0 C1 C2 C3 8E C5 C6 C7 EE
C9 CA CB CC 38 CE CF
D0 D1 D2 D3 3C D5 D6 D7 D8
D9 1F DB DC DD DE DF
E0 E1 E2 E3 E4 E5 E6 E7 E8
E9 EA EB EC ED 95 EF
F0 F1 F2 F3 15 F5 F6 F7 F8
4D FA FB FC FD FE FF
```

Example 2

This example shows a minor change in the message text.

This is a test message.

```
00000017    0083    00A8

D4 52 B6 03 04 A1 06 07 08
7A 0A 0B 0C 0D 0E 0F
A5 11 12 13 A6 F4 16 17 B5
19 1A 1B 1C 1D 1E DA
20 21 22 23 24 25 26 27 28
9B 2A 2B 2C 2D 2E 2F
30 31 32 33 BA 35 36 37 CD
39 3A 3B 00 3D 3E 3F
40 41 42 43 44 45 AE 47 48
49 4A 4B 4C F9 4E 4F
50 51 01 75 54 55 56 57 58
59 5A 5B 5C 5D 5E 5F
60 61 62 63 64 65 66 B8 68
69 6A 6B 6C 6D 6E 6F
70 71 72 73 74 53 76 77 78
79 09 7B 7C 7D 7E 7F
80 81 82 A8 84 85 86 87 88
89 8A 8B 8C 8D C4 8F
90 91 92 93 94 C8 96 97 98
99 9A 29 9C 9D 9E 9F
A0 05 A2 A3 A4 10 14 A7 18
A9 AA AB AC AD 46 AF
B0 B1 B2 B3 B4 83 02 B7 67
B9 34 BB BC BD BE BF
C0 C1 C2 C3 8E C5 C6 C7 EE
C9 CA CB CC 38 CE CF
D0 D1 D2 D3 3C D5 D6 D7 D8
D9 1F DB DC DD DE DF
E0 E1 E2 E3 E4 E5 E6 E7 E8
E9 EA EB EC ED 95 EF
F0 F1 F2 F3 15 F5 F6 F7 F8
4D FA FB FC FD FE FF
```

This is a text message.

```
00000017    00AB    00AD

D4 01 02 03 04 A1 52 6C 08
09 0A 0B 0C 0D 0E 0F
A5 11 1A 13 A6 15 16 17 B5
19 12 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28
9B 2A 2B 7F 2D 2E 2F
30 31 32 33 BA 35 36 37 CD
39 3A 3B 00 3D 3E 3F
40 41 42 43 44 45 46 47 48
49 4A 4B 4C 4D 4E 4F
50 51 59 53 54 55 56 57 7A
06 5A 5B 5C 5D 5E B3
60 61 62 63 64 65 66 67 68
69 6A 6B 73 6D 6E 6F
70 71 72 07 74 75 76 77 78
79 DF 7B 7C 7D 7E 2C
80 81 82 18 84 85 86 87 88
89 8A 8B 8C 8D C4 8F
90 91 92 93 94 EE 96 97 98
99 D2 29 9C 9D 9E 9F
A0 05 A2 A3 A4 10 14 A7 A8
A9 AA AD AC AB AE AF
B0 B1 B2 5F B4 83 B6 B7 B8
B9 34 BB BC BD BE BF
C0 C1 C2 C3 8E C5 C6 C7 C8
C9 CA CB CC 38 CE CF
D0 D1 9A D3 3C D5 D6 D7 D8
D9 DA DB DC DD DE 58
E0 E1 E2 E3 E4 E5 E6 E7 E8
E9 EA EB EC ED 95 EF
F0 F1 F2 F3 F4 F5 F6 F7 F8
F9 FA FB FC FD FE FF
```