To: T10 Technical Committee
From: Rob Elliott, HP (elliott@hp.com)
Date: 31 October 2003
Subject: 03-381r0 SBC-2 Data protection CRC seed

Revision history

Revision 0 (31 October 2003) First revision

Related documents

spc3r15 - SCSI Primary Commands - 3 revision 15
03-365 SPC-3 SBC-2 End-to-end data protection (George Penokie)

Overview

When data protection is added to tapes (SSC-3), it will have to accomodate variable length commands.

The CRC currently proposed for the DATA BLOCK GUARD field in 03-365 differs from the CRC used for Fibre Channel, SAS, etc. which all seed the CRC with 1s and transmit it inverted. The 03-365 CRC is seeded with 0s and transmitted uninverted.

If there are any leading 0s in the data, a 0 based seed does not detect the problem (XOR with 0 has no effect). If there are any trailing 0s in the data, a non-inverted CRC causes the same result. By seeding the CRC with 1 and inverting the CRC, those errors are very likely to be caught by the CRC.

Since the data protection CRC covers application data, extra 0s are more likely to appear than extra 1s, since software data buffers are often initialized to all 0s.

For disks with a fixed block size, this doesn't matter; if the data size is wrong, the data length of the command will show this. For tapes, however, the end of a file can appear at any offset, resulting in a short block that still needs to be covered.

The CRC for disks and tapes could be different - i.e., disks could seed with 0s and tapes with 1s. However, HBAs with hardware to generate the CRC on the fly would have to switch modes on a command-by-command basis. Normally they don't care whether they're talking to a disk or tape. It would be easier if all the uses of the CRC generator are identical.

There is one feature of the 0-based CRC that would be lost with a change: in a RAID set implementing RAID-5, the XOR of all the data drives' DATA BLOCK GUARD fields equals the parity drive's DATA BLOCK GUARD field. With a 1-based CRC, the parity drive contents (generated from the data drives) needs to be run through the CRC generator separately.

Proposal

Seed the CRC with all 1s and invert the CRC in the protection information.

Suggested changes to SBC-2 as modified by 03-365

4.5.3 Data block guard protection [replace entire section proposed by 03-365 section]

If data protection is enabled, the data block guard shall contain a CRC that is generated from the contents of the DATA BLOCK field.

Table 1 — CRC polynomials

Function	Definition
F(x)	A polynomial of degree k-1 that is used to represent the k bits of the data block covered by the CRC. For the purposes of the CRC, the coefficient of the highest order term shall be byte zero bit seven of the DATA BLOCK field.
L(x)	A degree 15 polynomial with all of the coefficients set to one: $L(x) = x^{15} + x^{14} + \ldots + x^2 + x^1 + 1$ (i.e., L(x) = FFFFh)
G(x)	The generator polynomial: $G(x) = x^{16} + x^{15} + x^{11} + x^9 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ (i.e., G(x) = 18BB7h)
R(x)	The remainder polynomial, which is of degree less than 16.
P(x)	The remainder polynomial on the receive checking side, which is of degree less than 16. P(x) = 0 indicates no error was detected.
Q(x)	The greatest multiple of G(x) in $(x^{16} \times F(x)) + (x^k \times L(x))$
Q'(x)	$x^{16} \times Q(x)$
M(x)	The sequence that is transmitted.
M'(x)	The sequence that is received.
C(x)	A unique polynomial remainder produced by the receiver upon reception of an error free sequence. This polynomial has the value: $C(x) = x^{16} \times \frac{L(x)}{G(x)}$ $C(x) = x^{15} + x^{13} + x^{11} + x^{10} + x^7 + x^5 + x^4$ (i.e., C(x) = ACB0h)

4.5.4 CRC generation

The equations that are used to generate the CRC from F(x) are as follows. All arithmetic is modulo 2.

CRC value = L(x) + R(x) = one's complement of R(x)

NOTE 1 - Adding L(x) (all ones) to R(x) produces the one's complement of R(x); this equation is specifying that the R(x) is inverted before it is transmitted.

The CRC is calculated by the following equation:

$$\frac{(x^{32} \times F(x)) + (x^K \times L(x))}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$$

The following equation specifies that the CRC is appended to the end of F(x):

$$M(x) = x^{32} \times F(x) + CRC$$

The bit order of F(x) presented to the CRC function is two bytes at a time starting with byte zero bit seven of the DATA BLOCK field until all the contents of the DATA BLOCK field are processed. An example of an even byte transfer is shown in figure 1.

[add inverters to output]

Figure 1 — Even byte CRC generator bit order

The received sequence M.(x) may differ from the transmitted sequence M(x) if there are transmission errors.

0.0.1 CRC checking

The received sequence M'(x) may differ from the transmitted sequence M(x) if there are transmission errors. The process of checking the sequence for validity involves dividing the received sequence by G(x) and testing the remainder. Direct division, however, does not yield a unique remainder because of the possibility of leading zeros. Thus a term L(x) is prepended to M'(x) before it is divided. Mathematically, the received checking is shown by the following equation:

$$x^{32} \times \frac{M'(x) + (x^{K} \times L(x))}{G(x)} = Q'(x) + \frac{P(x)}{G(x)}$$

In the absence of errors, the unique remainder is the remainder of the division as shown by the following equation:

$$\frac{P(x)}{G(x)} = x^{32} \times \frac{L(x)}{G(x)} = C(x)$$

The bit order of F(x) presented to the CRC checking function is the same order as the CRC generation bit order (see figure 75).

4.5.6 Test cases

Several test cases are shown in table 2.

Table 2 — CRC test cases

Pattern	CRC
32 bytes of 00h	DE47h
32 bytes of FFh	7CD4h
32 bytes of an incrementing pattern from 00h to 1Fh	DC63h
FFh, FFh, then 30 bytes of 00h	FFFFh
32 bytes of a decrementing pattern from FFh to E0h	7EF0h
16 bytes of 00h, then 32 bytes of an incrementing pattern from 00h to 1Fh	648Dh
32 bytes of 00h, then DE47h	ACB0h
32 bytes of FFh, then 7CD4h	ACB0h
32 bytes of an incrementing pattern from 00h to 1Fh, then DC63h	ACB0h

Annex C (informative) CRC

C.1 CRC implementation in C

The following is an example C program that generates the value for the DATA BLOCK GUARD field in protection information.

```
// dpcrc.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <stdio.h>
```

```
#include <malloc.h>

/* return crc value */
unsigned short calculate_crc(unsigned char *frame, unsigned long length) {
unsigned short const poly = 0x8BB7L;
unsigned const int poly_length = 16;
unsigned short crc_gen;
unsigned short x;
unsigned int i, j, fb;
unsigned const int invert = 1;/* 1=seed with 1s and invert the CRC */

crc_gen = 0x0000;
crc_gen ^= invert? 0xFFFF: 0x0000;   /* seed generator */

for (i = 0; i < length; i += 2) {
      /* assume little endian */
      x = (frame[i] << 8) | frame[i+1];
      printf ("x=%04x\n", x);

      /* serial shift register implementation */
        for (j = 0; j < poly_length; j++) {
            fb = ((x & 0x8000L) == 0x8000L) ^ ((crc_gen & 0x8000L) ==
0x8000L);
            x <<= 1;
            crc_gen <<= 1;
            if (fb)
                  crc_gen ^= poly;
      }
}

return crc_gen ^ (invert? 0xFFFF: 0x0000); /* invert output */
} /* calculate_crc */

/* function prototype */
unsigned short calculate_crc(unsigned char *, unsigned long);

void main (void) {
unsigned char *buffer;
unsigned long buffer_size = 32;
unsigned short crc;
unsigned int i;

/* all zeros */
buffer = (unsigned char *) malloc (buffer_size);
for (i = 0; i < buffer_size; i++) {
      buffer[i] = 0x00;
}
crc = calculate_crc(buffer, buffer_size);
printf ("Example CRC all-zeros is %04x\n", crc);
free (buffer);

/* all ones */
buffer = (unsigned char *) malloc (buffer_size);
for (i = 0; i < buffer_size; i++) {
      buffer[i] = 0xFF;
}
crc = calculate_crc(buffer, buffer_size);
```

```
    printf ("Example CRC all-ones is %04x\n", crc);
    free (buffer);

    /* incrementing */
    buffer = (unsigned char *) malloc (buffer_size);
    for (i = 0; i < buffer_size; i++) {
          buffer[i] = i;
    }
    crc = calculate_crc(buffer, buffer_size);
    printf ("Example CRC incrementing is %04x\n", crc);
    free (buffer);

    /* 4th pass */
    buffer = (unsigned char *) malloc (buffer_size);
    buffer[0] = 0xff;
    buffer[1] = 0xff;
    for (i = 2; i < buffer_size; i++) {
          buffer[i] = 0x00;
    }
    crc = calculate_crc(buffer, buffer_size);
    printf ("Example CRC 3 is %04x\n", crc);
    free (buffer);

    /* 5th pass */
    buffer = (unsigned char *) malloc (buffer_size);
    for (i = 0; i < buffer_size; i++) {
          buffer[i] = 0xff - i;
    }
    crc = calculate_crc(buffer, buffer_size);
    printf ("Example CRC 4 is %04x\n", crc);
    free (buffer);

    /* incrementing with leading 0s */
    buffer = (unsigned char *) malloc (buffer_size + 16);
    for (i = 0; i < 16; i++) {
          buffer[i] = 0x00;
    }
    for (i = 16; i < buffer_size + 16; i++) {
          buffer[i] = i - 16;
    }
    crc = calculate_crc(buffer, buffer_size + 16);
    printf ("Example CRC incrementing with leading 0s is %04x\n", crc);
    free (buffer);

    /* all zeros plus CRC */
    buffer = (unsigned char *) malloc (buffer_size + 2);
    for (i = 0; i < buffer_size; i++) {
          buffer[i] = 0x00;
    }
    buffer[32] = 0xde;
    buffer[33] = 0x47;
    crc = calculate_crc(buffer, buffer_size + 2);
    printf ("Example CRC all-zeros plus CRC is %04x\n", crc);
    free (buffer);

    /* all ones plus CRC */
    buffer = (unsigned char *) malloc (buffer_size + 2);
```

```
for (i = 0; i < buffer_size; i++) {
      buffer[i] = 0xff;
}
buffer[32] = 0x7c;
buffer[33] = 0xd4;
crc = calculate_crc(buffer, buffer_size + 2);
printf ("Example CRC all-ones plus CRC is %04x\n", crc);
free (buffer);

/* incrementing with CRC */
buffer = (unsigned char *) malloc (buffer_size + 2);
for (i = 0; i < buffer_size; i++) {
      buffer[i] = i;
}
buffer[32] = 0xdc;
buffer[33] = 0x63;
crc = calculate_crc(buffer, buffer_size + 2);
printf ("Example CRC incrementing plus CRC is %04x\n", crc);
free (buffer);

} /* main */
```