The iSCSI CRC32C Digest and the Simultaneous Multiply and Divide Algorithm

Luben Tuikov[*]
Splentec Ltd.
Richmond Hill, Ontario, Canada

Vicente Cavanna[†]
Agilent Technologies
Roseville, California, USA

January 30, 2002

Abstract

The CRC32C (aka CRC32/4) digest from iSCSI is presented in a rigorous algebraic manner, the why and how it works and the origin of its verifier constant. The most commonly used CRC digest computation algorithm in iSCSI and Ethernet, the Simultaneous Multiply and Divide (SMD), is derived from the long division algorithm. Sample implementations are provided of both algorithms.

0 Introduction

0.1 The generator polynomial $G(x)$ and arithmetic in $\mathbb{Z}_2$

iSCSI uses CRC32C digest which is defined by the generator polynomial represented in hexadecimal notation by `0x11edc6f41`, (see [1]), or in binary `100011110110111000110111101000001`. This defines $G(x)$, the generator polynomial as,

$$G(x) = x^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+x^{13}+x^{11}+x^{10}+x^{9}+x^{8}+x^{6}+1.$$

For convenience we will use the notation $G(x) = \sum_{i=0}^{n} g_i x^i$. Now this is a polynomial of degree 32, having 33 coefficients and thus taking exactly 33 bits to describe.

In this paper we will use n in place of 32, i.e. $n = \deg G(x)$, to make the discussion more general.

Please note that $G(x) \in \mathbb{Z}_2[x]$, as are all other polynomials used in this paper, unless otherwise stated. That is, the coefficients of $G(x)$ are in $\mathbb{Z}_2$. In other words, arithmetic operations on the coefficients are performed modulo 2. E.g.:

$$\left(x^7 + x^3 + 1\right) + \left(x^5 + x^3 + 1\right) = x^7 + x^5.$$

So, bitwise, $+$ and $-$ are the binary $\oplus$ (XOR) and multiplication is trivial.

[*]+1-905-707-1954x112, `luben@splentec.com` or `ltuikov@yahoo.com`.
[†]`vince_cavanna@agilent.com`.

0.2 The message

Let $\boldsymbol{\mu}$ be a sequence of bytes representing a message for which the CRC digest is sought. Let k be the number of bits in $\boldsymbol{\mu}$. Note that $k \equiv 0 \pmod 8$, since the message is an integer number of bytes. Furthermore let the largest store of $\boldsymbol{\mu}$ be a byte.[1] Then $\boldsymbol{\mu}$ is stored most significant byte, most significant bit[2] in memory.[3] Or, mathematically,

$$\boldsymbol{\mu} = \{b_i | i \in [0,k), b_i \in \mathbb{Z}_2, k \in \mathbb{N}\},$$

where bit b_i is more significant than bit b_{i+1}.

But this definition of a message does not yield to manipulation very easily, therefore we define

$$\mathcal{M} = \sum_{i=0}^{k-1} b_i x^{k-i-1}, \tag{1}$$

which is the polynomial representation of $\boldsymbol{\mu}$.

As per iSCSI, $\mathcal{M}$ is further described by the polynomial $M(x)$, of degree $k-1$,[4] where the coefficient to x^{k-1} is the least significant bit (bit 0) of the most significant byte, the coefficient to x^{k-2} is bit 1 of the most significant byte, and so on until the last bit, the coefficient to x^0 is the most significant bit of the least significant byte. I.e. the bytes are *mirrored.* To describe this mirroring mathematically, we can use the following transformation:[5]

$$T(i) = i + w - 1 - 2\,(i \bmod w)\,,$$

where $i \in \mathbb{N}$, w is the width of the largest store of the message (8, in our case), and arithmetic in $\mathbb{Z}$. Plugging in,

$$T(i) = i + 7 - 2\,(i \bmod 8)\,,$$

and this is the index to b, a bit in the sequence of bits in $\boldsymbol{\mu}$. Then $M(x)$ is described as follows,

$$M(x) = \sum_{i=0}^{k-1} b_{T(i)} x^{k-i-1}. \tag{2}$$

Please note that the difference between $\mathcal{M}$ and $M(x)$, is the order of bits in each byte of $\boldsymbol{\mu}$.

Lemma 1 *$T(T(i)) = i$, and thus $T^{-1}(i) = T(i)$.*

Proof: We proceed directly from the definition:

$$\begin{aligned}
T(T(i)) &= T(i) + w - 1 - 2(T(i) \bmod w) \\
&= i + w - 1 - 2(i \bmod w) + w - 1 - 2\left[(i + w - 1 - 2(i \bmod w)) \bmod w\right] \\
&= i + 2(w-1) - 2\left[(i \bmod w) + ((i + w - 1 - 2(i \bmod w)) \bmod w)\right] \\
&= i + 2(w-1) - 2\left[(2i + w - 1 - 2(i \bmod w)) \bmod w\right] \\
&= i + 2(w-1) - 2\left[(2i + w - 1 - 2i) \bmod w\right] \\
&= i + 2(w-1) - 2\left[(w-1) \bmod w\right] \\
&= i. \blacksquare
\end{aligned}$$

[1] As opposed to a *word*, or *double word*, etc.

[2] Without loss of generality (*wlg*).

[3] Padding bytes are set to 0 and included in $\boldsymbol{\mu}$.

[4] Without loss of generality, let's assume that the message starts (LSb of MSB) with a non-zero bit.

[5] The inquisitive reader may wish to show how $T(i)$ is derived.

Lemma 2 *$M(M(x)) = \mathcal{M}$, and thus $M(x) = M^{-1}(\mathcal{M})$.*

Proof: Using the result in Lemma 1,

$$M(M(x)) = \sum_{i=0}^{k-1} b_{T(T(i))} x^{k-i-1} = \sum_{i=0}^{k-1} b_i x^{k-i-1} = \mathcal{M}. \blacksquare$$

This lemma shows that applying $M(x)$ twice in succession has no effect.

1 Packet Sending

First let's define what a CRC digest is. A **CRC digest** is a transformation of the remainder of integer division of a polynomial by the generator polynomial. Note that the transformation should be chosen such that the degree of the CRC digest polynomial is less than that of the generator polynomial.

To calculate the CRC digest polynomial, we first[6] *augment* the message. That is, the message, $M(x)$, is multiplied by x^n. This is such that, later, when the CRC digest polynomial is computed it can be added to the dividend without changing the actual message data (bits).

In order to discern between messages of the same non-zero bits but starting with 0 or more zero bits (e.g. `0000101` and `101`), $M(x)$ is *prefixed* by a polynomial $D(x)$, where $D(x)$ is a non-zero polynomial and $\deg D(x) < n$.[7] That is, the coefficient to at least one of its terms is non-zero. We can express this as adding $C(x)$ to $x^n M(x)$, where,

$$C(x) = \sum_{i=0}^{n-1} d_i x^{n+k+i} = x^n x^k D(x) \tag{3}$$

This in fact means that a run of 1 or more zero bits at the beginning of the message is caught, since leading zeroes have no significance to the remainder, but a leading sequence of one or more 1's do.

So now we have,

$$M'(x) = x^n M(x) + C(x) = x^n \left(M(x) + x^k D(x) \right) = \sum_{i=0}^{k+2n-1} m'_i\, x^{k-i-1}. \tag{4}$$

And this clearly shows that we could have *prefixed* first and *then* multiplied by x^n.

Getting the remainder, $R(x)$, is trivial.[8] We have:

$$M'(x) = Q(x)G(x) + R(x) = Q(x)G(x) - R(x), \tag{5}$$

where $Q(x)$ is the quotient, $G(x)$ is the divisor, which is also the CRC generator polynomial, $R(x)$ is the remainder, and $M'(x)$ is the dividend. The equality above holds since the coefficients are in $\mathbb{Z}_2$.

Now we have to complement $R(x)$ as per iSCSI and add it to $M'(x)$. From Boolean algebra we know that $b \oplus 1 = \neg b$, then,

$$\neg R(x) = R(x) \oplus I(x) = R(x) + I(x), \tag{6}$$

[6] This can also be done second as we shall see shortly.

[7] The value of $D(x)$ and its significance will be discussed in section 4. The only requirement is that it is non-zero. WLG, assume that $\deg D(x) = n - 1$, this would make notation easier, and the logic is the same.

[8] Here you will have to compute it as per [2]. The algebraic implicit form is used here.

where

$$I(x) = x^{n-1} + x^{n-2} + \cdots + x^2 + x + 1. \tag{7}$$

Now add it, using (5) and (6) we have,

$$\begin{aligned} M''(x) &= M'(x) + \neg R(x) \\ &= Q(x)G(x) - R(x) + R(x) + I(x) \\ &= Q(x)G(x) + I(x), \end{aligned} \tag{8}$$

Then the message which is actually sent is (using (8) and (4)),

$$\begin{aligned} M'''(x) &= M''(x) - C(x) \\ &= M'(x) + \neg R(x) - C(x) \\ &= x^n M(x) + C(x) + \neg R(x) - C(x) \\ &= x^n M(x) + \neg R(x) \\ &= x^n M(x) + R(x) + I(x). \end{aligned} \tag{9}$$

That is, $x^n M(x) + \neg R(x)$ is sent to the receiver.

On the other end we receive $M'''(x)$, assuming there was no noise, and also noting that all link/TCP/Ethernet transformations are transparent to the sender and the recipient.

2 Packet Receiving

Now, let's investigate what happens to the message when it is received by an iSCSI implementation.

$M'''(x)$ is received and multiplied by x^n. Then $C'(x)$ is added to the result, where (see (3)),

$$C'(x) = x^n C(x) = x^{2n} x^k D(x), \tag{10}$$

since $\deg M'''(x) = n + k - 1$, (see (9)), then $\deg x^n M'''(x) = 2n + k - 1$, and $2n + k \leq \deg C'(x) \leq 3n + k - 1$. It should be clear from (4) and (9) that prefixing by $C(x)$ and then multiplying by x^n would give the same result.

So we have (using (9), (4), (5) and (8)),

$$\begin{aligned} M^{(4)}(x) &= C'(x) + x^n M'''(x) \\ &= C'(x) + x^n \left(x^n M(x) + R(x) + I(x)\right) \\ &= C'(x) + x^n \big(M'(x) - C(x)\big) + x^n \left(R(x) + I(x)\right) \\ &= x^n C(x) + x^n M'(x) - x^n C(x) + x^n \left(R(x) + I(x)\right) \\ &= x^n \big(M'(x) + R(x) + I(x)\big) \\ &= x^n \left(Q(x)G(x) - R(x) + R(x) + I(x)\right) \\ &= x^n M''(x) \\ &= x^n Q(x)G(x) + x^n I(x), \end{aligned} \tag{11}$$

equivalently,

$$M^{(4)}(x) \equiv x^n I(x) \pmod{G(x)}. \tag{12}$$

Since $\deg x^n I(x) > \deg G(x)$, $G(x)$ divides $x^n I(x)$, and we get,

$$x^n I(x) = Q'(x)G(x) + R'(x), \tag{13}$$

then substitute into (11) to get

$$\begin{aligned} M^{(4)}(x) &= x^n Q(x)G(x) + Q'(x)G(x) + R'(x) \\ &= \left(x^n Q(x) + Q'(x)\right) G(x) + R'(x). \end{aligned} \tag{14}$$

Clearly,

$$M^{(4)}(x) \equiv R'(x) \pmod{G(x)}. \tag{15}$$

Now it is worth to note that $R'(x)$ is the CRC digest polynomial for a received message. A recipient when calculating the CRC digest of a received message will get *that* polynomial. Now let's see what we know about $R'(x)$.

Using (12) and (15), we immediately see that,

$$R'(x) \equiv x^n I(x) \pmod{G(x)} \tag{16}$$

But x^n is known since n is known, $I(x)$ and $G(x)$ are also known, thus we can find the exact value of $R'(x)$ without having $M^{(4)}(x)$, and then check if we get the same value once we receive an actual message. Using (7),

$$x^n I(x) = x^{2n-1} + x^{2n-2} + \cdots + x^{n+1} + x^n,$$

plugging $n = 32$,

$$x^{32} I(x) = x^{63} + x^{62} + \cdots + x^{33} + x^{32}, \tag{17}$$

Remembering that all coefficients are in $\mathbb{Z}_2$ and using (16) and (17), we get

$$x^{32} I(x) \equiv R'(x) \pmod{G(x)} \tag{18}$$

which is exactly (13), but with $n = 32$. Performing the actual division we find that,

$$R'(x) = x^{28} + x^{27} + x^{26} + x^{21} + x^{19} + x^{18} + x^{16} + x^{12} + x^{11} + x^8 + x^7 + x^6 + x^5 + x^3 + x^2 + 1,$$

which is the *magic* value of `0x1c2d19ed` as required by iSCSI.

3 Implementation

This section gives the steps of a high-level implementation. Certain optimizations and detail is presented in section 4. The principles of bit shifting and table look-up are described in [2].

The peculiarity of iSCSI CRC, as well as Ethernet CRC is that bytes are mirrored (see section 0.2), when building the message polynomial $M(x)$.

We proceed as described in section 0.2: Let $\mathcal{M}$ be the message from which $M(x)$ is built. Furthermore let the largest store of $\mathcal{M}$ be a byte.[1] Then $\mathcal{M}$ is most significant byte, most significant bit[2] in memory.

An implementation proceeds as described in the previous sections:

1. Build $M(x)$ (see (2)), from $\mathcal{M}$, then form $M'(x)$ (see (4)).
2. Find the remainder, $R(x)$, of $\frac{M'(x)}{G(x)}$.
3. Complement $R(x)$, then mirror each byte of the complemented remainder.[9] Denote the result by $\overleftarrow{R(x)}$.

[9]Swap the bits inside the bytes. Do not change the byte positions.

4. Send $\mathcal{M}$ with $\overleftarrow{R(x)}$ appended at the end of the message (i.e. the last bits to send, starting with the coefficient to x^{31}, then the coefficient to x^{30}, all the way to the last bit, the coefficient to x^0 of $\overleftarrow{R(x)}$), noting that the transformations imposed by the link/Ethernet/TCP/IP layers are transparent to us. For convenience define $\mathcal{M}' = x^n\mathcal{M} + \overleftarrow{R(x)}$.

The message is received on the other end and there we proceed as follows:

1. Receive a message $\mathcal{M}'$, which is $\mathcal{M}$ with $\overleftarrow{R(x)}$ appended at the end as stipulated in step 4 above.

2. Building $M^{(4)}(x)$:

   (a) Mirror all bytes of $\mathcal{M}'$. We get $x^nM(x) + \neg R(x)$ (by construction of $\mathcal{M}'$, see step 4 above and section 0.2). Then multiplying by x^n we get (using (4) and (6)),

$$\begin{aligned} x^n\left(x^nM(x) + \neg R(x)\right) &= x^n\left(M'(x) - C(x)\right) + x^nR(x) + x^nI(x) \\ &= x^nM'(x) - x^nC(x) + x^nR(x) + x^nI(x). \end{aligned}$$

   (b) Add $C'(x)$ to get (using (10), (5) and the result above),

$$\begin{aligned} x^n\left(x^nM(x) + \neg R(x)\right) + C'(x) &= x^nM'(x) - x^nC(x) + x^nR(x) + x^nI(x) + C'(x) \\ &= x^nM'(x) + x^nR(x) + x^nI(x) \\ &= x^nQ(x)G(x) - x^nR(x) + x^nR(x) + x^nI(x) \\ &= x^nQ(x)G(x) + x^nI(x), \end{aligned}$$

   which is exactly (11), and thus equal to $M^{(4)}(x)$ of section 2.

3. Find the remainder modulo $G(x)$ (see [2]).

   As we saw in section 2, if the message is untainted, the bits of the remainder are equal to the *magic* value of `0x1c2d19ed`.

4 Optimization

In this section we show some algebraic equivalences which translate into direct optimization of the division process. For this we define two types of algorithms, **D** which is equivalent to **long division**, and another, **SMD**, to stand for **Simultaneous Multiply and Divide**, which we will introduce later as an optimization of **D**. Williams in [2] gives an informal introduction to **SMD** by use of look-up table. We will show how **SMD** is derived from **D** in a more formal manner. **SMD** is used in Ethernet and iSCSI implementations.

4.1 The D Algorithm

The **D** algorithm is essentially **long division**. It can be implemented in various ways, but most implementations use a *register* to keep the computed CRC digest after each byte or bit of the message has arrived. Long division natively processes the message bit at a time. To speed up this process, as new message bytes are processed, a table look-up can be used to get the new value to be XOR-ed with the CRC register. Williams in [2] shows how to compute and use a look-up table.

Since long division is a trivial algorithm to implement, in this sub-section will only show the equivalency between *prefixing* the message with some non-zero polynomial and *XOR-ing* the n most significant bits (MSb) of the message with another polynomial. These results will be helpful in *seeing the big picture* in the next sub-section where we show how the **SMD** algorithm is derived.

First let's examine (4),

$$\begin{aligned} M'(x) &= x^n M(x) + C(x) \\ &= x^n M(x) + x^n x^k D(x) \end{aligned}$$

which shows that a message is prefixed by the polynomial $D(x)$. But,

$$x^n D(x) \equiv E(x) \pmod{G(x)}. \tag{19}$$

That is, in modulo $G(x)$ arithmetic, $x^n D(x)$ is equivalent to some polynomial $E(x) = \sum_{i=0}^{n-1} e_i x^i$, which is the remainder of $\frac{x^n D(x)}{G(x)}$. But, $M'(x)$ *is* divided by $G(x)$ to find the remainder $R(x)$, thus,

$$\begin{aligned} M'(x) &= x^n M(x) + x^n x^k D(x) \\ &\equiv x^n M(x) + x^k E(x) \pmod{G(x)} \\ &= x^n \left(M(x) + x^{k-n} E(x) \right) \\ &= x^n \left(M(x) + \sum_{i=\max(0,n-k)}^{n-1} e_i x^{k-n+i} \right). \end{aligned} \tag{20}$$

And here we see that prefixing the message with $D(x)$ is equivalent to XOR-ing the n MSb of the message with the bits of $E(x)$.

When $k \gg n$, i.e. the number of message bits is a lot greater than the width of the CRC digest (n bits), then the above expression just XORs the bits of $E(x)$ with the n most significant bits of the message and *then* the message is multiplied by x^n.

When $k < n$, i.e. the message has less bits than the number of bits of the CRC, then we also XOR, but only the bits which make sense, and then the message is multiplied by x^n. Now let's see two examples.

4.1.1 Example: Prefixing the message by 32 1's

Let $D(x) = I(x)$, see (7) on page 4. This means that we *prefixed* the message with a string of 32 1's (bits with value 1). From (19),

$$E(x) = x^{32} I(x) \bmod G(x).$$

Performing the actual division we see that $E(x)$ is the bit pattern of the *magic* value `0x1c2d19ed`, see (18). This means that *prefixing* the message with 32 1's and then performing the division, is equivalent to XOR-ing the *magic* value with the 32 most significant bits of the message and then performing the division.[10]

[10]The repetitiveness of this sentence is intentional to show the algorithmic steps.

4.1.2 Example: Complementing the 32 most significant bits of the message

Let $E(x) = I(x)$. This would mean that we XOR the 32 most significant bits of the message with a string of 32 1's, effectively complementing the 32 most significant bits of the message. From (19), we see that

$$x^n D(x) \equiv I(x) \pmod{G(x)}.$$

This means that we are looking for a polynomial $D(x)$, such that, for some non-zero polynomial $K(x)$ the following holds,

$$\begin{aligned} x^n D(x) + I(x) &= K(x)G(x) \\ \underbrace{x^n D(x)}_{\deg \leq 2n-1} &= \underbrace{K(x)G(x) + I(x)}_{\deg \leq 2n-1}. \end{aligned} \tag{21}$$

We know that $\deg G(x) = n$ and $\deg I(x) = n - 1$, therefore $\deg K(x) \leq n - 1$. Also note that by multiplication, $x^n D(x)$, has its n lowest power terms equal to 0. This means that the n lowest power terms of $K(x)G(x)$ are all 1's. So, we are looking for a polynomial $K(x)$ of degree $n - 1$ or less, such that $K(x)G(x)$ has its n lowest degree terms equal to 1.

Note that $\deg(K(x)G(x)) \leq 2n - 1$, but we are interested only in the n lowest degree terms, so from the expansion of $K(x)G(x)$ we define the partial product sum,

$$\begin{aligned} P(x) &\overset{def}{=} \sum_{i=0}^{n-1} k_i x^i \left(\sum_{j=0}^{n-i-1} g_j x^j \right) \\ &= k_0 \left(\sum_{j=0}^{n-1} g_j x^j \right) + k_1 x \left(\sum_{j=0}^{n-2} g_j x^j \right) + k_2 x^2 \left(\sum_{j=0}^{n-3} g_j x^j \right) + \cdots + k_{n-1} x^{n-1} g_0. \end{aligned}$$

Expanding each term and collecting by x^p, where $p \in [0, n - 1]$,

$$\begin{aligned} P(x) &= x^0 (g_0 k_0) \\ &+ x^1 (g_1 k_0 + g_0 k_1) \\ &+ x^2 (g_2 k_0 + g_1 k_1 + g_0 k_2) \\ &\vdots \\ &+ x^{n-1} \left(\sum_{i=0}^{n-1} g_{n-1-i}\, k_i \right), \end{aligned} \tag{22}$$

and in general,

$$\begin{aligned} P(x) &= \sum_{j=0}^{n-1} x^j P_j(x), \text{ where} \\ P_j(x) &= \sum_{i=0}^{j} g_{j-i}\, k_i, \text{ and } j \in [0, n-1]. \end{aligned}$$

That is, each $P_j(x)$ is a coefficient to a term of $P(x)$ and we want $P_j(x) = 1$, $\forall j \in [0, n - 1]$.

Looking at (22), we see that this forms a linear system $\boldsymbol{GK} = \boldsymbol{P}$, where $\boldsymbol{P}$ is a column vector of n 1's, $\boldsymbol{K}$ is a column vector of the n coefficients of $K(x)$, which are being sought, and $\boldsymbol{G}$ is a lower triangular n by n matrix. $\boldsymbol{GK} = \boldsymbol{P}$ is

$$\begin{bmatrix} g_0 & 0 & 0 & \cdots & 0 \\ g_1 & g_0 & 0 & \cdots & 0 \\ g_2 & g_1 & g_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n-1} & g_{n-2} & g_{n-3} & \cdots & g_0 \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ \vdots \\ k_{n-1} \end{bmatrix} = \begin{bmatrix} g_0k_0 \\ g_1k_0 + g_0k_1 \\ g_2k_0 + g_1k_1 + g_0k_2 \\ \vdots \\ g_{n-1}k_0 + \cdots + g_0k_{n-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}. \tag{23}$$

Please note that if we had represented the vector $\boldsymbol{K}$ the other way around, k_{n-1} in row 0, k_{n-2} in row 1, etc., then $\boldsymbol{G}$ would be an upper triangular matrix formed from (22) similarly.[11]

This linear system has a solution $\boldsymbol{K} = \boldsymbol{G^{-1}P}$, where $\boldsymbol{G^{-1}}$ is the inverse of $\boldsymbol{G}$, and it exists since $g_0 \neq 0$.[12] $\boldsymbol{K}$ turns out to be `0x2914D53F`. Finding $K(x)G(x)$ is also trivial. We have, (*wlg*),

$$\begin{aligned} x^{32}D(x) &= K(x)G(x) + E(x) \\ &= \texttt{0x2a26f826FFFFFFFF} \oplus \texttt{0xFFFFFFFF} \\ &= \texttt{0x2a26f82600000000}. \end{aligned}$$

Thus, $D(x)$ is `0x2a26f826`.

That is, *complementing* the first 32 most significant bits of the message and then dividing, is equivalent to *prefixing* the message with `0x2a26f826` and then dividing.

Table 1 summarizes the results for both methods.

Method	Prefix equivalent	Adding to 32 MSb of $M(x)$ equivalent
Prefix by $D(x)$	$D(x)$	$E(x) = x^n D(x) \bmod G(x)$
Adding $E(x)$ to 32 MSb of $M(x)$	$D(x) = x^{-n}\left(K(x)G(x) + E(x)\right)$	$E(x)$

Table 1: Prefix⇔Complement Constant equivalency transformation table.

4.2 The SMD Algorithm

The **SMD** algorithm solves these problems:

- Having to multiply the message, $M(x)$, by x^n (see section 1).
- Having to *explicitly* prefix the message or to *explicitly* XOR the n MSb of the message by some non-zero polynomial.
- The need to keep the message in memory. Only $n+1$ bits of storage are required in order to compute the CRC digest of any finite length message. Moreover, the message length in bits doesn't have to have a factor of 8—we have the correct CRC after *each* message bit has been processed.

The **SMD** algorithm is essentially derived from the **D** algorithm and from equation (20),[13] and the fact that addition is associative in $\mathbb{Z}_2$. Williams in [2], gives examples of **D** in $\mathbb{Z}_2$, and an

[11] It is in fact the transpose of $\boldsymbol{G}$.

[12] $\boldsymbol{G^{-1}}$ is really easy to find and is left as an exercise to the reader.

[13] Note that **D** is a long division algorithm.

informal introduction to an **SMD** algorithm, with examples in the **C** programming language. Here we will show how the algorithm is derived.

In our case the divisor is $G(x)$, and the dividend is equation (20), in this form:

$$M'(x) = x^n \left(M(x) + \sum_{i=\max(0,n-k)}^{n-1} e_i\, x^{k-n+i} \right). \tag{24}$$

That is, the dividend is the message with its n MSb XOR-ed with $E(x)$, and n 0 bits added at the end of the message.

Now let's give some definitions. Let,

$$\begin{aligned} g(x) &\overset{def}{=} G(x) - g_n x^n = \sum_{i=0}^{n-1} g_i\, x^i, \\ W_i &\overset{def}{=} W_i(M'(x)) \overset{def}{=} \sum_{j=0}^{n} m'_{i+j}\, x^{n-j} \\ &\overset{def}{=} \sum_{j=0}^{n} w_{ij}\, x^j, \quad \text{where } i \in [0,k),\ \text{and } \deg W_i \le n,\ \forall i \in [0,k) \text{ and } w_{ij} = m'_{i+n-j}. \end{aligned}$$

That is, $g(x)$ is the n lower degree terms of $G(x)$ and W_i is a *window* of $n+1$ bits (as *wide* as $G(x)$), from bit i to bit $i+n$, of $M'(x)$ for the transmitter or of $M^{(4)}(x)$ for the receiver (see sections 0.2, 1 and 2). Note also that if the bytes were already mirrored, then we would not need the transformation $T(i+j)$ and instead would just use $i+j$.

Let's also define R_i to be the long division remainder after processing window W_i,

$$R_i \overset{def}{=} R_i(x) \overset{def}{=} \sum_{j=0}^{n-1} r_{i\,j} x^j, \text{ where } i \in [0,k),\ \text{and } \deg R_i < n,\ \forall i \in [0,k).$$

And let's define

$$d(A(x)) = a_n\,, \quad \text{where} \quad A(x) = \sum_{i=0}^{n} a_i x^i, \quad a_i \in \mathbb{Z}_2.$$

$d(A(x))$ is 0 or 1, its value depends on the coefficient to the x^n term of its polynomial argument.

Using these definitions, a **D** algorithm can be described as the following recurrence relation,[14]

$$\begin{aligned} R_0 &= W_0 + d(W_0)G(x)\,, \\ R_i &= xR_{i-1} + b_{T(n+i)} + d\left(xR_{i-1} + b_{T(n+i)}\right) G(x)\,, \quad \text{where} \quad i > 0. \end{aligned}$$

To simplify the equations and ease derivations, let

$$\begin{aligned} v_0 &= d(W_0)G(x)\,, \quad \text{and} \\ v_i &= d\left(xR_{i-1} + b_{T(n+i)}\right) G(x)\,, \end{aligned}$$

then the above recurrence relation becomes,

$$\boxed{\begin{aligned} R_0 &= W_0 + v_0\,, \\ R_i &= xR_{i-1} + b_{T(n+i)} + v_i\,, \quad \text{where} \quad i > 0\,. \end{aligned}} \tag{25}$$

[14]This is easy to see from long division.

Now let's expand it,

$$\begin{aligned}
R_0 &= W_0 + v_0. && W_0 \text{ processed.}\\
\\
R_1 &= xR_0 + b_{T(n+1)} + v_1\\
&= \underbrace{xW_0 + b_{T(n+1)}}_{W_1} + xv_0 + v_1\\
&= W_1 + xv_0 + v_1. && W_1 \text{ processed.}\\
\\
R_2 &= xR_1 + b_{T(n+2)} + v_2\\
&= \underbrace{xW_1 + b_{T(n+2)}}_{W_2} + x^2v_0 + xv_1 + v_2\\
&= W_2 + x^2v_0 + xv_1 + v_2. && W_2 \text{ processed.}\\
\\
R_3 &= xR_2 + b_{T(n+3)} + v_3\\
&= \underbrace{xW_2 + b_{T(n+3)}}_{W_3} + x^3v_0 + x^2v_1 + xv_2 + v_3\\
&= W_3 + x^3v_0 + x^2v_1 + xv_2 + v_3. && W_3 \text{ processed.}
\end{aligned}$$

And we see that in general, for $i > 0$,

$$\begin{aligned}
R_i &= W_i + x^iv_0 + x^{i-1}v_1 + \cdots + xv_{i-1} + v_i && (26)\\
R_{i+1} &= W_{i+1} + x^{i+1}v_0 + x^iv_1 + \cdots + xv_i + v_{i+1}\\
&= W_{i+1} + x\left(x^iv_0 + x^{i-1}v_1 + \cdots + xv_{i-1} + v_i\right) + v_{i+1}\\
&= W_{i+1} + x\left(R_i - W_i\right) + v_i. && (27)
\end{aligned}$$

Clearly from the last two results,

$$R_i - W_i = x\left(R_{i-1} - W_{i-1}\right) + v_i, \quad \text{where} \quad i > 0.$$

Now if we define,

$$S_i = R_i - W_i\,, \tag{28}$$

then using (25), (26) and the last definition, we can rewrite our last result as,

$$\begin{aligned}
S_0 &= v_0\,,\\
S_i &= xS_{i-1} + v_i\,, \quad i > 0.
\end{aligned} \tag{29}$$

Let's substitute v_i, use (28) and expand it,

$$\begin{aligned}
S_i &= xS_{i-1} + d\left(xR_{i-1} + b_{T(n+i)}\right)G(x)\\
&= xS_{i-1} + d\left(xS_{i-1} + \underbrace{xW_{i-1} + b_{T(n+i)}}_{W_i}\right)G(x)\\
&= xS_{i-1} + d\left(xS_{i-1} + W_i\right)G(x).
\end{aligned}$$

That is, at each step we can only compute S_i and should we need to know the remainder we can use (28) to get it,

$$\begin{aligned}
R_i &= S_i + W_i\\
&= xS_{i-1} + W_i + d\left(xS_{i-1} + W_i\right)G(x).
\end{aligned}$$

Please note here that $\deg R_i < n$ no matter what xS_{i-1} and W_i are.

Now let's investigate what happens when the last data of the message is processed, namely W_{k-1}, and what the remainder would be,

$$\begin{aligned} R_{k-1} &= S_{k-1} + W_{k-1} \\ &= xS_{k-2} + W_{k-1} + d\,(xS_{k-2} + W_{k-1})\,G(x). \end{aligned}$$

Since, by the definition (25), R_{k-1} will always have n or less terms, we can ignore all coefficients of the x^n terms on the right side of the equation,[15] namely for xS_{k-2}, W_{k-1} and $G(x)$. Now if we define

$$\bar{d}(A(x)) = A(x) - a_n x^n\,, \quad \text{where} \quad A(x) = \sum_{i=0}^{n} a_i x^i, \quad a_i \in \mathbb{Z}_2,$$

to yield only the lower n terms of its polynomial argument, then we can rewrite the above result as follows,

$$R_{k-1} = \bar{d}(xS_{k-2}) + \underbrace{\bar{d}(W_{k-1})}_{0} + d\,(xS_{k-2} + W_{k-1})\,\underbrace{\bar{d}(G(x))}_{g(x)}.$$

So instead of $G(x)$, we can use $g(x)$, instead of W_{k-1} we can use $W_{k-1} - w_{k-1\,n}x^n$, and similarly, instead of xS_{k-1} we can use $xS_{k-1} - s_{k-1\,n-1}x^n$. But $W_{k-1} - w_{k-1\,n}x^n$ is zero since the message was multiplied by x^n; that is, W_{k-1} is the last $n+1$ data bits of the message, from bit $k-1$ (which is the last message bit) to bit $k-1+n$ (the last augmented message bit), but the last n bits are 0 since they came to be when the message was multiplied by x^n. Also $d(W_i)$ is just $b_{T(i)}$, and $d(xS_{k-2} + W_{k-1}) = d(xS_{k-2}) + d(W_{k-1}) = d(xS_{k-2}) + b_{T(k-1)}$, which means the most significant bit of xS_{k-2} (which is also the MSb of just S_{k-2}) XOR-ed with the most significant bit of W_{k-1}. Therefore we need not multiply the message by x^n. Rewriting it,

$$R_{k-1} = \bar{d}(xS_{k-2}) + \big(d\,(xS_{k-2}) + b_{T(k-1)}\big)\,g(x).$$

Which is equivalent to S_{k-1},

$$S_{k-1} = \bar{d}(xS_{k-2}) + \big(d\,(xS_{k-2}) + b_{T(k-1)}\big)\,g(x).$$

That is, as long as $k > 0$, i.e. the message exists, then S_{k-1} is the remainder of the *augmented* (i.e. multiplied by x^n) such message. And this is the algorithm which you will most likely find in Ethernet and iSCSI.

Implementors prefer to keep $\bar{d}(xS_{k-2})$ in a *register* of n bits (32 in our case) and we need one more bit to store $b_{T(i)}$, thus we need a total of $n+1$ (i.e. 33) bits to compute the remainder after each bit has arrived.

Now let's investigate the initial conditions and the apparent XOR-ing of an initial non-zero polynomial with the first n bits of the message. Until now we used W_0 to indicate the first $n+1$ bits of the message, but since we are XOR-ing the first n bits with $E(x)$ (see (24)) we have to reflect that. From (25), we now see that,

$$\begin{aligned} R_0 &= xE(x) + W_0 + v_0\,, \quad \text{and from (28)}\,, \\ S_0 &= xE(x) + v_0 \\ &= xE(x) + d\,(W_0 + xE(x))\,G(x) \\ &= xE(x) + \big(d(xE(x)) + b_{T(0)}\big)\,G(x). \end{aligned}$$

[15]By construction of definition (25), after addition, the coefficient of the x^n term will be 0.

Ignoring the coefficient to the highest power term, the remainder of an augmented k bit pure message[16] can then be computed as follows,

$$\boxed{\text{The } \mathbf{SMD} \text{ Algorithm}}$$
$$\boxed{\begin{array}{lrcl} k=1: & S_0 & = & \bar{d}\,(xE(x)) + \big(d(xE(x)) + b_{T(0)}\big)\, g(x)\,, \\ k>1: & S_{k-1} & = & \bar{d}\,(xS_{k-2}) + \big(d(xS_{k-2}) + b_{T(k-1)}\big)\, g(x). \end{array}} \tag{30}$$

This shows how one can compute the remainder of $M'(x)$ described in (24), by only having $M(x)$ and $E(x)$. That is, the message is *not* altered in any way ($M(x)$, only byte-mirrored), but the result of applying **SMD** to the message, is the remainder one would get if they had multiplied the message by x^n *and* either prefixed the message with $D(x)$, or added $E(x)$ to the n MSb of $M(x)$, and then applied the **D** algorithm (long division), see Table 1 on page 9.

Notice that the index means which bit number in the sequence of message bits is being processed and is remnant of the definition of R_i. If we define A_i to be the **SMD** remainder after processing i bits then (30) can be expressed as,

$$\begin{array}{rrcl} 0 \text{ bits processed:} & A_0 & = & E(x) \\ i>0 \text{ bits processed:} & A_i & = & \bar{d}(xA_{i-1}) + \big(d(xA_{i-1}) + b_{T(i-1)}\big)\, g(x). \end{array} \tag{31}$$

The most common CRC digest computation algorithms implement *exactly* (30) or the equivalent (31) and now we will show how simply it translates into an implementation.

First note that since $\deg E(x) < n$, it has an n bit representation and can be represented by an n bit *register* `A`. $xE(x)$ simply means left shift by 1 bit, and $\bar{d}(xE(x))$ simply means to ignore the newly created from the left shift $e_{n-1}x^n$ term, just like in an n bit register (by $\bar{d}()$ which was defined previously). $d(xE(x))$ is just the coefficient to the x^n term which is e_{n-1} (by $d()$ which was defined previously). And similarly for the S_{k-2} term.

Let `A` be a n bit register, let `MSb()` yield the most significant bit of its argument (cf. $d()$), and let `LeftShift1()` yield the left shift by one bit, of its argument, and XOR be the exclusive-or operator. Then the **SMD** algorithm described by (30) or equivalently by (31) in pseudo-programming language is exactly this:

```
RESULT = 0
A = E(x)
WHILE THERE ARE MORE MESSAGE BITS
    b = next message bit (byte mirrored)
    IF  b XOR MSb(A)  THEN
        A = LeftShift1(A) XOR g(x)
    ELSE
        A = LeftShift1(A)
    RESULT = A
END WHILE
```

If the message data is available byte at a time and an implementation has $32n$ bytes available, then a look-up table can be constructed and the above algorithm can be sped-up to process byte at a time rather than bit at a time. Formally, each entry in the table has n bits and can be found by expressing the value of `A` after 8 iterations of the above algorithm; there are at most 2^8 entries. Williams in [2] gives an informal introduction to the table look-up **SMD** algorithm, how to construct the table and how to use it.

[16]That is, an augmented pure message of k bits has $k+n$ bits. And it is pure, since its n MSb are not XOR-ed with any polynomial.

5 Conclusion

In this paper we showed how the iSCSI CRC32C works and why it works in a rigorous mathematical manner. The origin of the *magic* value `0x1c2d19ed` was also shown.

In section 3 we gave the steps for a sample implementation, with respect to both the sender and the receiver.

The computation of CRC digests often includes transformations of the data for which the digest is sought, like *prefixing* the message with some non-zero polynomial and *suffixing* the message with a string of zero bits. Some implementations will not have the available memory or other resources to perform these transformations, in which case they will have to resort to different algorithms to compute the CRC digest.

In section 4 we derived the Simultaneous Multiply and Divide (**SMD**) algorithm from the classical prefix-suffix-divide algorithm. We also showed the transformations which map a polynomial which is being prefixed to another polynomial suitable to be XOR-ed with the n most significant bits of the message (and the reverse transformation), while upon division the digest is the same.

SMD *eliminates* the following:

- The need to keep the message in memory.
- The need to suffix the message by a string of zero bits.
- The need to prefix the message or to *explicitly* XOR the n most significant bits of the message by some non-zero polynomial.
- The need to know the message length. The correct CRC digest is obtained after each message bit is processed for the current length of the message.

That is, computing the digest of a k-bit long message ($k > 0$) with the **SMD** algorithm yields the same digest as prefixing and suffixing the k-bit message and then performing division by $G(x)$. An implementation of the **SMD** algorithm was also shown in pseudo-programming language.

Symbols

$G(x)$ The generator polynomial. **1**

n The degree of $G(x)$. It is also the width, or the number of bits of the remainder.

k The number of bits in the message. **2**

$\boldsymbol{\mu}$ The message as a sequence of bytes. **2**

$\mathcal{M}$ The message as a polynomial. **2**

$T : \mathbb{N} \to \mathbb{N}$ The byte-mirroring transformation. **2**

$M(x)$ The byte-mirrored message. **2**

$D(x)$ A non-zero polynomial. $M(x)$ is prefixed by this polynomial, such that leading zero-valued bits are considered part of the message when the remainder is computed. **3**

$C(x)$ A polynomial composed of $D(x)$ in such a way, that when added to the augmented message, we get the *prefixed* augmented message. **3**

$M'(x)$ The prefixed, augmented message. That is, the result of prefixing the message by $D(x)$ and multiplying it by x^n (appending n 0 valued bits at the end). This is the input to a long division algorithm (the **D** algorithm). **3**

$R(x)$ The remainder of $M'(x)$ when divided by $G(x)$. **3**

$I(x)$ Identity polynomial of degree $n-1$. **4**

$M'''(x)$ The message, including the original message bits, which is sent to the receiver. This is used in the theory part of the discussion and byte-mirroring is implicit. **4**

$M^{(4)}$ The prefixed and augmented message as received by the receiver. **4**

$R'(x)$ The remainder of $M^{(4)}$ when divided by $G(x)$. This is also the *magic value*. **5**

$\overleftarrow{R(x)}$ The complemented and then byte-mirrored $R(x)$. **5**

$\mathcal{M}'$ The data bits which are sent to the receiver; explicit byte mirroring. This is an implementation definition. **6**

D algorithm This is the classical long division algorithm. **6**

$E(x)$ The remainder of $x^n D(x)$ when divided by $G(x)$. **7**

SMD algorithm This is the Simultaneous Multiply Divide algorithm. **9**

$g(x)$ This is $G(x) - g_n x^n$, i.e. same as $G(x)$ but can "fit" in an n bit register. **10**

W_i A polynomial of degree less than or equal to n. This is a *window* of $n+1$ bits from bit i to bit $i+n$ of a prefixed, augmented and byte-mirrored message. **10**

R_i The remainder of dividing a prefixed, augmented and byte-mirrored message by $G(x)$ after processing W_i. **10**

$d(A(x))$ This yields the value of the coefficient to the x^n term of $A(x)$, which can be 0 or 1. **10**

$\bar{d}(A(x))$ This yields only the lower n terms of $A(x)$, i.e. $a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$. **12**

Thanks

Thanks to Paul Koning of EqualLogic for initially pointing out that once the message has been *received* it is *again* multiplied by x^{32} and *then* the CRC digest is calculated.

References

[1] Castagnoli, et al. *Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits*, G. Castagnoli, S. Bräuer, M. Herrmann, IEEE Transactions on Communications, Vol. 41, No. 6, June 1993, pp. 883-892.

[2] Ross N. Williams. *A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS*, 1993.
`ftp://ftp.adelaide.edu.au/pub/rocksoft/crc_v3.txt`
`ftp://ftp.rocksoft.com/papers/crc_v3.txt`
`http://www.repairfaq.org/filipg/LINK/F_crc_v3.html`.