The dvdisaster RS03 Reed-Solomon Codec specification

Carsten Gnörlich
carsten@dvdisaster.org

Version 1.00

first draft

Abstract

This paper describes the data format of the dvdisaster RS03 Reed-Solomon codec. The codec creates Reed-Solomon parity data to protect data stored on optical media. Parity data can either be stored in a separate file or be integrated with the .iso image on the same medium. While these features were already present in the preceeding codecs RS01 and RS02, the new RS03 codec will be fully multi-threadable. It can therefore take full advantage of multicore processor systems and is expected to become the default codec soon after its introduction in dvdisaster 0.80 (see `http://dvdisaster.org` for an overview of the dvdisaster project).

Target audience. This paper is primarily intended as a working base for the dvdisaster developers and, when the final version has been crafted, as an implementation guide for third party developers who wish to create and process dvdisaster RS03 data. It is **neither intended nor suitable** as end-user documentation; for usage information please refer to the online documentation at `http://dvdisaster.org`.

Prerequisites. This paper assumes profound knowledge of coding theory and the underlying math. The reader is assumed to have a thorough understanding of Reed-Solomon codes, both in theory and from an implementation viewpoint. A basic understanding of programming in C is also assumed.

Copyright 2008-2010 Carsten Gnörlich. Verbatim copying and distribution of this entire article is permitted in any medium, provided this notice is preserved.

1 Changelog

V1.00 Clarified: RS03 header does not contain copy of first CRC sector (appendix A).
Added $sectorsPerLayer$ field in Ecc header and CRC block format.
Added ecc file specification.

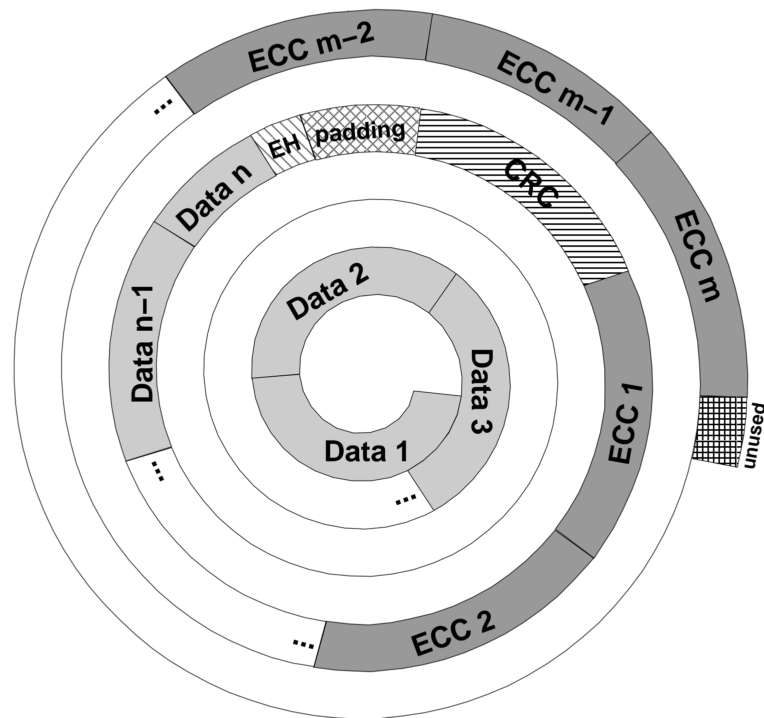

Figure 1: Physical RS03 layout

2 The RS03 data layout

2.1 Physical layout

Optical media are recorded as a single long spiral[1] of sectors which are indexed beginning with 0. The first sector lies at the innermost position of the spiral and numbering continues onward to the outside of the spiral.
Reed-Solomon encoding works best when errors are evenly distributed over all ecc blocks. Therefore we must strive to spread our ecc blocks evenly over the media surface. To facilitate such distribution, dvdisaster logically divides the medium into 255 units which are called "layers" for historical reasons. Figure 1 illustrates how a medium is divided into n data layers, one CRC layer, and m ecc layers, with $n + m + 1 = 255$. Ecc blocks are comprised by taking one byte from each layer as shown in fig. 2 on the following page. This distributes the ecc block reasonably good over the medium surface.
Layer size is measured in numbers of sectors which are 2048 bytes in size. All 255 layers have the same size. The data layers map exactly to the iso image which is to be protected by dvdisaster; e.g. the number and sequence of sectors in $Data_1, \ldots, Data_n$ is the same as in the iso image. Two extra sectors are appended to the ISO image holding the ecc header "EH"; these are logically treated as a part of the ISO image. If the ISO image size plus the two EH headers is not an integer multiple of the layer size, the last (n-th) data layer will be padded accordingly. The data layers are followed by a CRC layer. Each CRC layer sector contains a data structure holding CRC32 checksums for the data sectors plus additional parameters which were used dur-

[1]Multiple layered media contain one spiral for each physical layer, but are otherwise conceptually identical.

Figure 2: Logical RS03 layout

ing the RS03 encoding process. The data and CRC layers are protected by the Reed-Solomon parity which is stored in the remaining layers ($ECC_1, \ldots, ECC_m$). Since the medium capacity is not necessarily an integer multiple of 255, some unused sectors remain at the end of the medium. These are neither written nor referenced in any way.

The RS03 data can be stored either directly on the medium or into a separate file. Figure 1 shows the first case where ecc data is embedded into the image; this is also called a “RS03 augmented ISO image” in dvdisaster terminology. In the other case a separate error correction file will be created containing the sectors starting with the EH header. The following discussion is based on the augmented image case; see section 2.7 for handling the file based format.

2.2 Logical layout

The relationship between layers and ecc blocks is stated again in the logical view presented in figure 2. Here the 255 layers are shown in stacked order. From each layer the $i-th$ byte corresponds to the $i-th$ error correction block. The parity is calculated using n data bytes $d_{i,1}, \ldots, d_{i,n}$ plus the crc byte c_i. The resulting m roots of the completed Reed-Solomon code are then stored in the ecc bytes $e_{i,1}, \ldots, e_{i,m}$.
Since the input iso image plus the two EH sectors is usually not an integer multiple of the layer size, the last data layer $data_n$ may contain padding sectors containing a special signature. The content of the padding sectors is used in the ecc byte calculation and is written into the augmented image.

Medium type	Maximum size	Layer size
CD	359.424	1.409
DVD 1 layer	2.295.104	9.000
DVD 2 layers	4.171.712	16.359
BD 1 layer	11.826.176	46.377
BD 2 layers	23.652.352	92.754

Table 1: Sector size parameters for several media types

2.3 Calculating the layout for encoding

When encoding with RS03 the layout of the augmented image is fully specified by two values: the maximum medium size and the iso image size (from now on, “size” always means “number of 2048K sectors”).

Media sizes are hard coded and taken from table 1. Since we need to divide the medium into 255 layers, the layer size is:

$$layer\ size = \left\lfloor \frac{medium\ size}{255} \right\rfloor$$

This allows us to compute the number of data layers needed to cover the iso image plus the ecc header:

$$data\ layers = \left\lceil \frac{iso\ image\ size + 2}{layer\ size} \right\rceil$$

The number of padding sectors in the last data layer is:

$$padding\ sectors = layer\ size * data\ layers - iso\ image\ size - 2$$

For Reed-Solomon encoding, we will have to encode

$$n\ data\ bytes = data\ layers + 1$$

and produce the following number of parity bytes:

$$m\ roots = 255 - n\ data\ bytes$$

The RS03 augmented image must fill the medium completely (except for the $medium\ size \bmod 255$ sectors at the end). However for performance reasons the maximum redundancy is capped to 200%, or 170 roots. This means that the ISO image must at least span the first 255-170=85 layers, otherwise additional padding will be added to fill up the 85 data layers. This situation is not reflected in the calculations and figure shown above.

2.4 Re-calculating the layout from defective media

In order to recover a defective medium, the values of *layer size* and *data layers* need to be determined. The RS03 format allows for three heuristics with increasing complexity for learning about these values:

2.4.1 Using the Ecc Header

All required information can be obtained from the data structures of the Ecc Header which is described in appendix A. If ecc data is stored in a separate error correction file, the first 4096 bytes of the ecc file yield the Ecc Header. Otherwise, let n be the size of the ISO file system which can be obtained from the ISO file system master block. Then the Ecc Header is typically found in the RS03-augmented image at sectors $n, n+1$ or at $n+150, n+151$ (due to padding inserted by some popular CD-R mastering software).

If the ISO file system master block is unreadable, the Ecc Header can be identified by its characteristic signature and checksum. If the Ecc Header is encountered during reading of the defective medium it might be worthwhile to generate a tentative ISO master block in the image file. This would speed up future processing of the image; however current implementations of dvdisaster do not yet implement this feature.

2.4.2 Using the CRC layer

Each CRC layer sector contains a data structure which not only holds the CRC32 checksums but also a copy of important parameters from the Ecc header (see section 2.5 for details). CRC sectors can be easily recognized by looking for their signature and checksum while scanning the medium image. If dvdisaster finds a valid CRC sector and the Ecc header is defective, a tentative Ecc header is written to the image to speed up further operations on the image file.

However it should be noted that since all CRC sectors are stored consecutively on the medium, they can easily be wiped out by a large defective region on the medium. Therefore, another heuristic exists for learning about the RS03 layout.

2.4.3 Evaluating the Reed-Solomon code

If neither the Ecc Header nor any CRC sectors are readable the RS03 layout can be determined by the following heuristic.

First, the medium size is determined from table 1. This is always possible as long as the drive will recognize the medium at all. Since the layer size is $\left\lfloor \frac{medium\ size}{255} \right\rfloor$, the location of the 255 layers on the medium is now known. The remaining task is to find out the redundancy of the Reed-Solomon code, e.g. how many layers contain roots for the RS code.

Taking the *i-th* sector from each layer will produce a valid error correction block, but with unknown redundancy. As RS03 will create redundancies using 8 to 170 roots, we employ a brute-force approach by evaluating the Reed-Solomon code for $8..170$ roots. If the error correction is successful for n roots and the sector from layer *255-1-n* yields the CRC data structure, the correct number of roots has been found.

In reality, not all 162 combinations of roots need to tested since additional information can be exploited:

1. If the sector from layer *255-1-n* is present/readable, we do not need to test for n roots any further: Encoding with n roots would have produced a CRC sector in this place.

2. If the number of erasures (as indicated by unreadable sectors) is higher than n, we can trivially skip the RS decoding. We might have to test another set of 255 sectors though if testing for all other numbers of roots fails as well.

Criterion 1) should quickly narrow down the possible numbers of roots in the average case, e.g. when enough redundancy is available for recovering the medium. Worst case behaviour of trying each ecc block for 8..170 roots is likely to appear only when the medium is unrecoverable, e.g. when more sectors are damaged than the Reed-Solomon code can correct.

2.5 Contents of the CRC layer

Each sector of the CRC layer contains the data structure shown in appendix B. Following the numbering from figure 2, CRC sector c_i contains the CRC32 checksums for data sectors $d_{j,1}, \ldots, d_{j,n}$ with $j = (i+1) \bmod \mathit{layer\ size}$. The purpose of this offset is to have the error correction of ECC block i recover the CRC checksum for the next ECC block $i+1$. In case of readable but corrupted sectors this will keep the error correction in erasure mode and therefore save precious redundancy (the RS code can recover twice as much errors when the location of defective data is known).

Checksums for data sector $d_{j,k}$ are stored in array element CrcBlock->crc[k]. Unused array elements are set to zero. The remaining contents of the CRC sector structure provide configuration and layout information; see appendix B for details.

2.6 Encoding the ecc layers

Encoding the error correction information requires reading and buffering of at least 255 sectors comprising the ecc block (see fig. 2 for a definition of the ecc block). A possible encoding algorithm might process each ecc block at a time. For each ecc block i it would do the following:

First, the n data sectors $d_{i,1}, \ldots, d_{i,n}$ of the ecc block are read in. The CRC layer sector c_i is initialized, filled in with checksums generated by processing the previous ecc block, and completed by calculating its own checksum *selfCRC*. Unused portions of c_i remain zero. Afterwards the CRC32 checksums of $d_{i,1}, \ldots, d_{i,n}$ are calculated and stored away using the same buffering mechanism. Since the hand-over of CRC checksums between ecc blocks is the only place where RS03 does not fully parallelize, data sector I/O and CRC32 caching needs to be carefully thought out in multithreaded implementations.

Once $d_{i,1}, \ldots, d_{i,n}$ and c_i have been prepared, 2048 sets of a RS(255,k) code (with $k = 255 - n - 1$) are calculated by looping over the 2048 bytes of the ecc sectors. If l denotes a certain byte position between $0, \ldots, 2047$ in the ecc block sectors, then the *l-th* byte from $d_{i,1}, \ldots, d_{i,n}, c_i$ is retrieved and fed into the RS(255,k) encoder. The resulting parity bytes $p_1, \ldots, p_m$ are stored in byte position l of the ecc layer sectors $e_{i,1}, \ldots, e_{i,m}$. When all 2048 bytes of the ecc block sectors have been processed the ecc layer sectors can be written out; either into the error correction file or into the RS03 augmented image.

The RS(255,k) encoder is the same for RS01, RS02 and RS03. See appendix C for the parameters used in the encoder.

2.7 Encoding as a separate error correction file

If the image size is too close to the medium capacity, not enough space is left for augmenting the image with redundancy. dvdisaster will refuse to augment images when there is insufficient

space for at least 8 roots. Creating images with less than 43 roots (20% of redundancy) will trigger a warning that the error correction capacity may be too low. In those cases, storing the error correction information in a separate file comes as an alternative.

RS03 error correction files ("ecc files") contain the same error correction information and layout as in the augmented image case, with the following differences:

Omittance of data padding sectors. While the image format shown in figures 1 and 2 may contain padding sectors between the ecc header and the CRC layer, those sectors are not written into the ecc file. The padding sectors are however required during encoding and decoding, e.g. they need to be virtually created in memory when processing the respective ecc blocks. Therefore an ecc file providing *nroots* of redundancy will contain *2 + (nroots+1) * layer size* sectors. Physically it will contain the ecc header, then the CRC layer and finally the *nroots* ecc layers.

Freely chooseable redundancy. In the augmented image case the redundancy is always chosen to fill up the medium completely. For ecc files the redundancy can be freely chosen by the user between 8 roots (3.2%) and 170 roots (200%). Encoding with more than 170 roots is technically possible, but run-time requirements get out of proportion; hence the selectable redundancy is capped at 200%.

As a consequence of the variable redundancy the ecc file layout can only be determined by looking at the ecc header or CRC sectors. The strategy of experimentally evaluating the Reed-Solomon code (see sub section 2.4.3) however can not be applied to ecc files since neither the size of the padding area nor the original size of the possibly truncated image and ecc files can be determined.

To see whether this is really a limiting factor we look at the typical outcome of recovering a single file from a defective medium:

- The ecc file is fully read, but random sectors are damaged.
- The ecc file is truncated to the position of the first read error.

In both scenarios it is highly likely that at least one CRC sectors survives at the beginning of the file; in that case the error correction will not only recover the image but also repair the ecc file into its original state.

Although this gives RS03 ecc files good chances to remain functional even when being partly damaged, it is highly recommended to store ecc files only on media which are themself being protected by dvdisaster. ISO and UDF file systems do not have sufficient redundancy for their meta data (e.g. directory structures). If such meta data becomes unreadable a significant number of files may become completely inaccessible. Please note that this is a general weakness of file-based data protection and recovery: The meta data is not part of any file and can therefore not be protected by any error correction data put inside the file(s). This is the also the simple reason why we did not use tools like PAR2 and developed dvdisaster instead; the image-based approach of dvdisaster protects both files and meta data.

A Ecc header format

The ecc header is defined in the include file *dvdisaster.h*. Its C definition is as follows:

```
typedef struct _EccHeader
{   gint8 cookie[12];            /* "*dvdisaster*" */
    gint8 method[4];             /* e.g. "RS01" */
    gint8 methodFlags[4];        /* 0-2 for free use by the respective methods; 3 see above */
    guint8 mediumFP[16];         /* fingerprint of FINGERPRINT SECTOR */
    guint8 mediumSum[16];        /* complete md5sum of whole medium */
    guint8 eccSum[16];           /* md5sum of ecc code section of .ecc file */
    guint8 sectors[8];           /* number of sectors medium is supposed to have */
    gint32 dataBytes;            /* data bytes per ecc block */
    gint32 eccBytes;             /* ecc bytes per ecc block */
    gint32 creatorVersion;       /* which dvdisaster version created this */
    gint32 neededVersion;        /* oldest version which can decode this file */
    gint32 fpSector;             /* sector used to calculate mediumFP */
    guint32 selfCRC;             /* CRC32 of EccHeader (currently RS02 only) – since V0.66 –*/
    guint8 crcSum[16];           /* md5sum of crc code section of RS02 .iso file */
    gint32 inLast;               /* bytes contained in last sector */
    guint64 sectorsPerLayer;     /* layer size for RS03 */
    gint8 padding[3976];         /* pad to 4096 bytes: room for future expansion */
} EccHeader;
```

The ecc header is used in all ecc formats (RS01, RS02, RS03) of dvdisaster, but not all fields apply to all formats. See the following table for the meaning and usage of the fields:

Field	Usage	Format(s)
cookie	Magic byte sequence for recognizing the header. Contains the string `*dvdisaster*`.	all
method	4 characters describing the format; currently allowed: `RS01`, `RS02`, `RS03`.	all
methodFlags	4 bytes for further specification of the format.	
	Byte 0 contains the following flag: Bit 0 - The *mediumSum* field is valid. Bit 1 - Set to 1 in ecc files.	 RS03 RS03
	Bytes 1-2 are unused in the current methods.	
	Byte 3 contains the following flags: Bit 0 - ecc data was created by a development release. Bit 1 - ecc data was created by a release candidate. If these bits are present, the user will be hinted that he is using ecc data from a non-stable dvdisaster version.	all

(continued on next page)

$mediumFP$	The md5sum of the sector specified by the *fpSector*. The sector should be chosen to have a huge probability being unique to the medium; currently sector 16 (the ISO filesystem root sector) is used.	all
$mediumSum$	The md5sum of the ISO image. For RS01 this is the md5sum of the whole image; for RS02 it is calculated for the original ISO image (without the added RS02 sectors). RS03 uses this value only when bit 1 in *methodFlags* is set.	all
$eccSum$	On RS01 this is the md5sum of the ecc file excluding the first 4096 bytes. For RS02 this is the md5sum calculated over the md5sums of the $nroots$ ecc layers. RS03 does not use this value.	RS01, RS02
$sectors$	For error correction files this is the number of sectors in the protected medium. If augmented images are used, this denotes the number of sectors in the original ISO image (without the added RS02/RS03 sectors).	all
$dataBytes$	The number of data layers, including the CRC layer.	all
$eccBytes$	The number of ecc layers (= number of roots) for the parity. $dataBytes + eccBytes = 255$.	all
$creatorVersion$	The dvdisaster version used for creating this ecc data. A decimal value 102345 would mean dvdisaster version 10.23.45.	all
$neededVersion$	The minimum dvdisaster version required for processing this ecc data. Version encoding as above.	all
$fpSector$	The sector used for calculating $mediumFP$.	all
$selfCRC$	A CRC32 checksum of the ecc header itself. Not used header fields are set to zero and the selfCRC field is initialized to the value 0x4c5047 (little endian).	
$crcSum$	md5sum of the CRC layer in RS02 encoded images.	RS02
$inLast$	The number of Bytes contained in the last image sector. This allows for encoding of files with arbitrary length, not just ISO images. dvdisaster versions prior to V0.66 do not use this field and always assume it to be 2048 which is the default for iso images.	all
$sectorsPerLayer$	The number of sectors per layer.	RS03
$padding$	The ecc header is zero padded to a length of 4096 bytes. Future codes may allocate additional space for the zero padding. See the note below for usage of the upper 2048 bytes on RS02/RS03.	all
Byte 2048-4096	A copy of the first CRC layer sector.	RS02

B CRC block format

The crc layer contains 2048 byte blocks containing the data structure described below. Except for the CRC32 checksums most of the information contained in this data structure is copied from the Ecc Header described in appendix A. The crc block format is defined in the include file *dvdisaster.h* and has the following C definition:

```
typedef struct _CrcBlock
{   guint32 crc[256];            /* Checksum for the data sectors */
    gint8 cookie[12];            /* "*dvdisaster*" */
    gint8 method[4];             /* e.g. "RS03" */
    gint8 methodFlags[4];        /* 0-2 for free use by the respective methods; 3 see above */
    gint32 creatorVersion;       /* which dvdisaster version created this */
    gint32 neededVersion;        /* oldest version which can decode this file */
    gint32 fpSector;             /* sector used to calculate mediumFP */
    guint8 mediumFP[16];         /* fingerprint of FINGERPRINT SECTOR */
    guint8 mediumSum[16];        /* complete md5sum of whole medium */
    guint64 dataSectors;         /* number of sectors of the payload (e.g. iso file sys) */
    gint32 inLast;               /* bytes contained in last sector */
    gint32 dataBytes;            /* data bytes per ecc block */
    gint32 eccBytes;             /* ecc bytes per ecc block */
    guint64 sectorsPerLayer;     /* for recalculation of layout */
    guint32 selfCRC;             /* CRC32 of ourself, zero padded to 2048 bytes */
} CrcBlock;
```

The CrcBlock data structure is used in the CRC layer of RS03 augmented images only. RS02 has a similar CRC layer but uses a different concept for retrieving layout information from the image. The following table describes the meaning and usage of the CrcBlock fields:

Field	Usage
crc	If this data structure is found in the *i-th* sector of the CRC layer, it contains the CRC32 checksum for data sectors $d_{j,1}, \ldots, d_{j,n}$, with $j = (i+1) \bmod \mathit{layer\ size}$. See figure 2 for details. Please note that the crc[] array is filled starting from crc[0], and unused field are left zero.
cookie	Magic byte sequence for recognizing the header. Contains the string `*dvdisaster*`.
method	4 characters describing the format; currently only "RS03" may appear here.

(continued on next page)

$methodFlags$	4 bytes for further specification of the format. Byte 0 contains the following flags: Bit 0 - The *mediumSum* field is valid. Bit 1 - Set to 1 in ecc files. Bytes 1-2 are unused in the current methods. Byte 3 contains the following flags: Bit 0 - ecc data was created by a development release. Bit 1 - ecc data was created by a release candidate. If these bits are present, the user will be hinted that he is using ecc data from a non-stable dvdisaster version.
$creatorVersion$	The dvdisaster version used for creating this ecc data. A decimal value 102345 would mean dvdisaster version 10.23.45.
$neededVersion$	The minimum dvdisaster version required for processing this ecc data. Version encoding as above.
$fpSector$	The sector used for calculating $mediumFP$.
$mediumFP$	The md5sum of the sector specified by the *fpSector*. The sector should be chosen to have a huge probability being unique to the medium; currently sector 16 (the ISO filesystem root sector) is used.
$mediumSum$	The md5sum of the original ISO image if the first bit in the *methodFlags* field is set. Since md5sum generation can not be parallelized, the user may opt not to calculate this checksum if multi core encoding is used.
$dataSectors$	For error correction files this is the number of sectors in the protected medium. If augmented images are used, this denotes the number of sectors in the original ISO image (without the added padding and RS03 sectors).
$inLast$	The number of Bytes contained in the last image sector. This allows for encoding of files with arbitrary length, not just ISO images.
$dataBytes$	The number of data layers, including the CRC layer.
$eccBytes$	The number of ecc layers (= number of roots) for the parity. $dataBytes + eccBytes = 255$.
$sectorsPerLayer$	The number of sectors per layer.
$selfCRC$	A CRC32 checksum of the ecc header itself. Not used fields are set to zero and the selfCRC field is initialized to the value 0x4c5047 (little endian).
remaining bytes	The CrcBlock is zero padded to a size of 2048 bytes.

C RS(255,k) encoding parameters and examples

dvdisaster uses a standard, non-shortened Reed-Solomon code with the following commonly used parameters:

The Galois field tables are generated by the field generator polynomial 0x187 ($1 + X + X^2 + X^7 + X^8$). The Reed-Solomon code generator polynomial is created using element 0x70 as first consecutive root and the primitive element 0x0b.

As a starting point for testing your own implementation, some values and tables are shown below. The logarithm and anti-logarithm tables in the Galois field are shown in tables 2 and 3. Please note that there is no need for hard-coding these tables as their contents can be enumerated by using the field generator polynomial.

When encoding for 32 roots, the RS code generator polynomial will be:
01 5b 7f 56 10 1e 0d eb 61 a5 08 2a 36 56 ab 20 71 20 ab 56 36 2a 08 a5 61 eb 0d 1e 10 56 7f 5b 01
or in index form:
00 f9 3b 42 04 2b 7e fb 61 1e 03 d5 32 42 aa 05 18 05 aa 42 32 d5 03 1e 61 fb 7e 2b 04 42 3b f9 00

Using the above generator polynomial for encoding the data byte sequence $\{0, 1, \ldots, 222\}$ produces the following parity bytes:
2f bd 4f b4 74 84 94 b9 ac d5 54 62 72 12 ee b3 eb ed 41 19 1d e1 d3 63 20 ea 49 29 0b 25 ab cf

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	ff	00	01	63	02	c6	64	6a	03	cd	c7	bc	65	7e	6b	2a
10	04	8d	ce	4e	c8	d4	bd	e1	66	dd	7f	31	6c	20	2b	f3
20	05	57	8e	e8	cf	ac	4f	83	c9	d9	d5	41	be	94	e2	b4
30	67	27	de	f0	80	b1	32	35	6d	45	21	12	2c	0d	f4	38
40	06	9b	58	1a	8f	79	e9	70	d0	c2	ad	a8	50	75	84	48
50	ca	fc	da	8a	d6	54	42	24	bf	98	95	f9	e3	5e	b5	15
60	68	61	28	ba	df	4c	f1	2f	81	e6	b2	3f	33	ee	36	10
70	6e	18	46	a6	22	88	13	f7	2d	b8	0e	3d	f5	a4	39	3b
80	07	9e	9c	9d	59	9f	1b	08	90	09	7a	1c	ea	a0	71	5a
90	d1	1d	c3	7b	ae	0a	a9	91	51	5b	76	72	85	a1	49	eb
a0	cb	7c	fd	c4	db	1e	8b	d2	d7	92	55	aa	43	0b	25	af
b0	c0	73	99	77	96	5c	fa	52	e4	ec	5f	4a	b6	a2	16	86
c0	69	c5	62	fe	29	7d	bb	cc	e0	d3	4d	8c	f2	1f	30	dc
d0	82	ab	e7	56	b3	93	40	d8	34	b0	ef	26	37	0c	11	44
e0	6f	78	19	9a	47	74	a7	c1	23	53	89	fb	14	5d	f8	97
f0	2e	4b	b9	60	0f	ed	3e	e5	f6	87	a5	17	3a	a3	3c	b7

Table 2: Galois field logarithm table

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	01	02	04	08	10	20	40	80	87	89	95	ad	dd	3d	7a	f4
10	6f	de	3b	76	ec	5f	be	fb	71	e2	43	86	8b	91	a5	cd
20	1d	3a	74	e8	57	ae	db	31	62	c4	0f	1e	3c	78	f0	67
30	ce	1b	36	6c	d8	37	6e	dc	3f	7e	fc	7f	fe	7b	f6	6b
40	d6	2b	56	ac	df	39	72	e4	4f	9e	bb	f1	65	ca	13	26
50	4c	98	b7	e9	55	aa	d3	21	42	84	8f	99	b5	ed	5d	ba
60	f3	61	c2	03	06	0c	18	30	60	c0	07	0e	1c	38	70	e0
70	47	8e	9b	b1	e5	4d	9a	b3	e1	45	8a	93	a1	c5	0d	1a
80	34	68	d0	27	4e	9c	bf	f9	75	ea	53	a6	cb	11	22	44
90	88	97	a9	d5	2d	5a	b4	ef	59	b2	e3	41	82	83	81	85
a0	8d	9d	bd	fd	7d	fa	73	e6	4b	96	ab	d1	25	4a	94	af
b0	d9	35	6a	d4	2f	5e	bc	ff	79	f2	63	c6	0b	16	2c	58
c0	b0	e7	49	92	a3	c1	05	0a	14	28	50	a0	c7	09	12	24
d0	48	90	a7	c9	15	2a	54	a8	d7	29	52	a4	cf	19	32	64
e0	c8	17	2e	5c	b8	f7	69	d2	23	46	8c	9f	b9	f5	6d	da
f0	33	66	cc	1f	3e	7c	f8	77	ee	5b	b6	eb	51	a2	c3	00

Table 3: Galois field anti-logarithm table