Raptor Forward Error Correction Scheme
draft-ietf-rmt-bb-fec-raptor-object-01

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on December 25, 2005.

Copyright Notice

Copyright (C) The Internet Society (2005).

Abstract

This document describes an FEC scheme for the Raptor forward error correction code and its application to reliable delivery of data objects.
Raptor is a fountain code, i.e., as many encoding symbols as needed can be generated by the encoder on-the-fly from the source symbols of a source block of data. The decoder is able to recover the source block from any set of encoding symbols only slightly more in number than the number of source symbols.

The Raptor code described here is a systematic code, meaning that the first encoding symbols generated are equal to the source symbols.
# Table of Contents

1. Introduction ........................................... 4
2. Requirements notation ................................. 5
3. Formats and Codes ..................................... 6
   3.1 FEC Payload IDs .................................. 6
   3.2 FEC Object Transmission Information ............... 6
      3.2.1 Mandatory .................................. 6
      3.2.2 Common ..................................... 6
      3.2.3 Scheme-Specific .............................. 7
4. Procedures ............................................. 8
   4.1 Content Delivery Protocol Requirements ............. 8
   4.2 Example parameter derivation algorithm .......... 8
5. Raptor FEC code specification ........................ 11
   5.1 Introduction ..................................... 11
   5.2 Definitions, Symbols and abbreviations .......... 11
      5.2.1 Definitions .................................. 11
      5.2.2 Symbols ..................................... 12
      5.2.3 Abbreviations ................................ 14
   5.3 Overview .......................................... 14
   5.4 Object delivery .................................... 15
      5.4.1 Source block construction ...................... 15
      5.4.2 Encoding packet construction ................. 17
   5.5 Systematic Raptor encoder ........................ 18
      5.5.1 Encoding overview .............................. 18
      5.5.2 First encoding step: Intermediate Symbol Generation .... 19
      5.5.3 Second encoding step: LT encoding ............. 24
      5.5.4 Generators .................................. 24
   5.6 Example FEC decoder ............................... 27
      5.6.1 General ..................................... 27
      5.6.2 Decoding a source block ....................... 27
   5.7 Random Numbers ..................................... 32
      5.7.1 The table V0 ................................ 32
      5.7.2 The table V1 ................................ 33
   5.8 Systematic Indices J(K) ............................ 34
6. Security Considerations ............................... 47
7. IANA Considerations ................................... 48
8. Intellectual Property .................................. 49
9. Acknowledgements ..................................... 50
10. References ............................................ 50

1. Introduction

This document specifies an FEC Scheme for the Raptor forward error correction code for object delivery applications. The concept of an FEC Scheme is defined in [2] and [4].

The Raptor FEC Scheme is a Fully-Specified FEC Scheme.

Raptor is a fountain code, i.e., as many encoding symbols as needed can be generated by the encoder on-the-fly from the source symbols of a block. The decoder is able to recover the source block from any set of encoding symbols only slightly more in number than the number of source symbols.

The code described in this document is a systematic code, that is, the original source symbols are sent unmodified from sender to receiver, as well as a number of repair symbols.

The code described here is identical to that described in [5].
2. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [1].
3. Formats and Codes

3.1 FEC Payload IDs

The FEC Payload ID MUST be a 4 octet field defined as follows:

+---------------------------+---------------------------+
| Source Block Number (SBN) | Encoding Symbol ID (ESI)  |
+---------------------------+---------------------------+

Figure 1: FEC Payload ID format

Source Block Number (SBN), (16 bits): An integer identifier for the source block that the encoding symbols within the packet relate to.

Encoding Symbol ID (ESI), (16 bits): An integer identifier for the encoding symbols within the packet.

The interpretation of the Source Block Number and Encoding Symbol Identifier is defined in Section 5.

3.2 FEC Object Transmission Information

3.2.1 Mandatory

The value of the FEC Encoding ID MUST be <tbd>.

3.2.2 Common

The Common FEC Object Transmission Information elements used by this FEC Scheme are:

- Transfer Length (F)
- Encoding Symbol Length (T)

The Transfer Length is a non-negative integer less than 2^^48. The Encoding Symbol Length is a non-negative integer less than 2^^16.

Where a Content Delivery Protocol requires an encoding for these elements to be specified by the FEC Scheme then the Transfer Length MUST be encoded as a 6 octet field in network byte order (high order byte first) and the Encoding Symbol Length MUST be encoded as a 2 octet field in network byte order (high order byte first).
3.2.3 Scheme-Specific

The following parameters are carried in the Scheme-Specific FEC Object Transmission Information element for this FEC Scheme:

- The number of source blocks (Z)
- The number of sub-blocks (N)
- A symbol alignment parameter (Al)

These parameters are all non-negative integers and MUST be encoded as a 4-octet field as defined in Figure 2.

```
 3 2 1
10987654321098765432109876543210
+-------------------------+-------+-------+
|    Z     |   N   |   Al  |
+-------------------------+-------+-------+
```

Figure 2: Encoding of Scheme-specific FEC Object Transmission Information
4. Procedures

4.1 Content Delivery Protocol Requirements

This section describes the information exchange between the Raptor FEC Scheme and any Content Delivery Protocol (CDP) making use the Raptor FEC Scheme for object delivery.

The Raptor encoder and decoder for object delivery require the following information from the CDP:

- The transfer length of the object, F, in bytes
- The symbol alignment parameter, Al
- The symbol size, T, in bytes, which MUST be a multiple of Al
- The number of source blocks, Z
- The number of sub-blocks in each source block, N

The Raptor encoder for object delivery additionally requires:

- the object to be encoded, F bytes

The Raptor encoder supplies the CDP with encoding packet information consisting, for each packet, of:

- Source Block Number (SBN)
- Encoding Symbol ID (ESI)
- Encoding symbol(s)

The CDP MUST communicate this information transparently to the Raptor decoder. A suitable CDP is defined in [3].

4.2 Example parameter derivation algorithm

This section provides recommendations for the derivation of the four transport parameters, Al, T, Z and N. This recommendation is based on the following input parameters:

- F the object size, in bytes
- W a target on the sub-block size, in bytes
the maximum packet payload size, in bytes, which is assumed to be a multiple of Al

- Al the symbol alignment factor, in bytes

- Kmax the maximum number of source symbols per source block.

Note: Section 5.2.2 defines Kmax to be 8192.

- Kmin a minimum target on the number of symbols per source block

- Gmax a maximum target number of symbols per packet

Based on the above inputs, the transport parameters T, Z and N are calculated as follows:

Let,

\[ G = \min(\lceil P\cdot K_{\text{min}} / F \rceil, P / Al, G_{\text{max}}) \]

\[ T = \lfloor P / (Al\cdot G) \rfloor \cdot Al \]

\[ K_t = \lceil F / T \rceil \]

\[ Z = \lceil K_t / K_{\text{max}} \rceil \]

\[ N = \min(\lceil \lceil K_t / Z \rceil \cdot T / W \rceil, T / Al) \]

The values of G and N derived above should be considered as lower bounds. It may be advantageous to increase these values, for example to the nearest power of two. In particular, the above algorithm does not guarantee that the symbol size, T, divides the maximum packet size, P, and so it may not be possible to use the packets of size exactly P. If, instead, G is chosen to be a value which divides P/Al, then the symbol size, T, will be a divisor of P and packets of size P can be used.

The algorithm above and that defined in Section 5.4.1.2 ensure that the sub-symbol sizes are a multiple of the symbol alignment parameter, Al. This is useful because the XOR operations used for encoding and decoding are generally performed several bytes at a time, for example at least 4 bytes at a time on a 32 bit processor. Thus the encoding and decoding can be performed faster if the sub-symbol sizes are a multiple of this number of bytes.

Recommended settings for the input parameters, Al, Kmin and Gmax are as follows: Al = 4, Kmin = 1024, Gmax = 10.
The parameter W can be used to generate encoded data which can be decoded efficiently with limited working memory at the decoder. Note that the actual maximum decoder memory requirement for a given value of W depends on the implementation, but that it is possible to implement decoding using working memory only slightly larger than W.
5. Raptor FEC code specification

5.1 Introduction

This Section specifies the Raptor forward error correction code and its application to object delivery. Raptor is a fountain code, i.e., as many encoding symbols as needed can be generated by the encoder on-the-fly from the source symbols of a block. The decoder is able to recover the source block from any set of encoding symbols only slightly more in number than the number of source symbols.

The code described in this document is a systematic code, that is, the original source symbols are sent unmodified from sender to receiver, as well as a number of repair symbols.

5.2 Definitions, Symbols and abbreviations

5.2.1 Definitions

For the purposes of this specification, the following terms and definitions apply.

Source block: a block of K source symbols which are considered together for Raptor encoding purposes.

Source symbol: the smallest unit of data used during the encoding process. All source symbols within a source block have the same size.

Encoding symbol: a symbol that is included in a data packet. The encoding symbols consist of the source symbols and the repair symbols. Repair symbols generated from a source block have the same size as the source symbols of that source block.

Systematic code: a code in which the source symbols are included as part of the encoding symbols sent for a source block.

Repair symbol: the encoding symbols sent for a source block that are not the source symbols. The repair symbols are generated based on the source symbols.

Intermediate symbols: symbols generated from the source symbols using an inverse encoding process. The repair symbols are then generated directly from the intermediate symbols. The encoding symbols do not include the intermediate symbols, i.e., intermediate symbols are not included in data packets.
Symbol: a unit of data. The size, in bytes, of a symbol is known as the symbol size.

Encoding symbol group: a group of encoding symbols that are sent together, i.e., within the same packet whose relationship to the source symbols can be derived from a single Encoding Symbol ID.

Encoding Symbol ID: information that defines the relationship between the symbols of an encoding symbol group and the source symbols.

Encoding packet: data packets that contain encoding symbols.

Sub-block: a source block is sometimes broken into sub-blocks, each of which is sufficiently small to be decoded in working memory. For a source block consisting of K source symbols, each sub-block consists of K sub-symbols, each symbol of the source block being composed of one sub-symbol from each sub-block.

Sub-symbol: part of a symbol. Each source symbol is composed of as many sub-symbols as there are sub-blocks in the source block.

Source packet: data packets that contain source symbols.

Repair packet: data packets that contain repair symbols.

5.2.2 Symbols

i, j, x, h, a, b, d, v, m represent positive integers.

ceil(x) denotes the smallest positive integer which is greater than or equal to x.

choose(i,j) denotes the number of ways j objects can be chosen from among i objects without repetition.

floor(x): denotes the largest positive integer which is less than or equal to x.

i % j denotes i modulo j.

X ^ Y denotes, for equal-length bit strings X and Y, the bitwise exclusive-or of X and Y.
\( A_l \) denotes a symbol alignment parameter. Symbol and sub-symbol sizes are restricted to be multiples of \( A_l \).

\( A \) denotes a matrix over GF(2).

\( \text{Transpose}[A] \) denotes the transposed matrix of matrix \( A \)

\( A^{^-1} \) denotes the inverse matrix of matrix \( A \)

\( K \) denotes the number of symbols in a single source block

\( K_{\text{max}} \) denotes the maximum number of source symbols that can be in a single source block. Set to 8192.

\( L \) denotes the number of pre-coding symbols for a single source block

\( S \) denotes the number of LDPC symbols for a single source block

\( H \) denotes the number of Half symbols for a single source block

\( C \) denotes an array of intermediate symbols, \( C[0], C[1], C[2], \ldots, C[L-1] \)

\( C' \) denotes an array of source symbols, \( C'[0], C'[1], C'[2], \ldots, C'[K-1] \)

\( X \) a non-negative integer value

\( V_0, V_1 \) two arrays of 4-byte integers, \( V_0[0], V_0[1], \ldots, V_0[255] \) and \( V_1[0], V_1[1], \ldots, V_1[255] \)

\( \text{Rand}[X, i, m] \) a pseudo-random number generator

\( \text{Deg}[v] \) a degree generator

\( \text{LTEnc}[K, C, (d, a, b)] \) a LT encoding symbol generator

\( \text{Trip}[K, X] \) a triple generator function

\( G \) the number of symbols within an encoding symbol group

\( \text{GF}(n) \) The Galois field with \( n \) elements.

\( N \) the number of sub-blocks within a source block
\[ T \] the symbol size in bytes. If the source block is partitioned into sub-blocks, then \( T = T' \times N \).

\[ T' \] the sub-symbol size, in bytes. If the source block is not partitioned into sub-blocks then \( T' \) is not relevant.

\[ F \] the object size, for object delivery, in bytes

\[ I \] the sub-block size in bytes

\[ P \] for object delivery, the payload size of each packet, in bytes, that is used in the recommended derivation of the object delivery transport parameters.

\[ Q \] \( Q = 65521 \), i.e., \( Q \) is the largest prime smaller than \( 2^{16} \)

\[ Z \] the number of source blocks, for object delivery

\( J(K) \) the systematic index associated with \( K \)

\[ G \] denotes any generator matrix

\[ I_S \] denotes the \( S \times S \) identity matrix

\[ 0_{S \times H} \] denotes the \( S \times H \) zero matrix

\( a^{^b} \) \( a \) raised to the power \( b \)

5.2.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

- **ESI** Encoding Symbol ID
- **LDPC** Low Density Parity Check
- **LT** Luby Transform
- **SBN** Source Block Number
- **SBL** Source Block Length (in units of symbols)

5.3 Overview

The principle component of the systematic Raptor code is the basic
encoder described in Section 5.5. First, it is described how to derive values for a set of intermediate symbols from the original source symbols such that knowledge of the intermediate symbols is sufficient to reconstruct the source symbols. Secondly, the encoder produces repair symbols which are each the exclusive OR of a number of the intermediate symbols. The encoding symbols are the combination of the source and repair symbols. The repair symbols are produced in such a way that the intermediate symbols and therefore also the source symbols can be recovered from any sufficiently large set of encoding symbols.

This document defines the systematic Raptor code encoder. A number of possible decoding algorithms are possible. An efficient decoding algorithm is provided in Section 5.6.

The construction of the intermediate and repair symbols is based in part on a pseudo-random number generator described in Section 5.5.4.1. This generator is based on a fixed set of 512 random numbers which MUST be available to both sender and receiver. These are provided in Section 5.7.

Finally, the construction of the intermediate symbols from the source symbols is governed by a 'systematic index', values of which are provided in Section 5.8 for source block sizes from 4 source symbols to \( K_{\text{max}} = 8192 \) source symbols.

5.4 Object delivery

5.4.1 Source block construction

5.4.1.1 General

In order to apply the Raptor encoder to a source object, the object may be broken into \( Z \geq 1 \) blocks, known as source blocks. The Raptor encoder is applied independently to each source block. Each source block is identified by a unique integer Source Block Number (SBN), where the first source block has SBN zero, the second has SBN one, etc. Each source block is divided into a number, \( K \), of source symbols of size \( T \) bytes each. Each source symbol is identified by a unique integer Encoding Symbol Identifier (ESI), where the first source symbol of a source block has ESI zero, the second has ESI one, etc.

Each source block with \( K \) source symbols is divided into \( N \geq 1 \) sub-blocks, which are small enough to be decoded in the working memory. Each sub-block is divided into \( K \) sub-symbols of size \( T' \).

Note that the value of \( K \) is not necessarily the same for each source
block of a object and the value of $T'$ may not necessarily be the same for each sub-block of a source block. However, the symbol size $T$ is the same for all source blocks of a object and the number of symbols, $K$ is the same for every sub-block of a source block. Exact partitioning of the object into source blocks and sub-blocks is described in Section 5.4.1.2 below.

5.4.1.2 Source block and sub-block partitioning

The construction of source blocks and sub-blocks is determined based on five input parameters, $F$, $Al$, $T$, $Z$ and $N$ and a function $\text{Partition}[]$. The five input parameters are defined as follows:

- $F$ the transfer length of the object, in bytes
- $Al$ a symbol alignment parameter, in bytes
- $T$ the symbol size, in bytes, which MUST be a multiple of $Al$
- $Z$ the number of source blocks
- $N$ the number of sub-blocks in each source block

These parameters MUST be set so that $\text{ceil}(\text{ceil}(F/T)/Z) \leq K_{\text{max}}$. Recommendations for derivation of these parameters are provided in Section 4.2.

The function $\text{Partition}[]$ takes a pair of integers $(I, J)$ as input and derives four integers $(IL, IS, JL, JS)$ as output. Specifically, the value of $\text{Partition}[I, J]$ is a sequence of four integers $(IL, IS, JL, JS)$, where $IL = \text{ceil}(I/J)$, $IS = \text{floor}(I/J)$, $JL = I - IS \times J$ and $JS = J - JL$. $\text{Partition}[]$ derives parameters for partitioning a block of size $I$ into $J$ approximately equal sized blocks. Specifically, $JL$ blocks of length $IL$ and $JS$ blocks of length $IS$.

The source object MUST be partitioned into source blocks and sub-blocks as follows:

Let,

$$K_t = \text{ceil}(F/T)$$

$$(KL, KS, ZL, ZS) = \text{Partition}[K_t, Z]$$

$$(TL, TS, NL, NS) = \text{Partition}[T/Al, N]$$

Then, the object MUST be partitioned into $Z = ZL + ZS$ contiguous source blocks, the first $ZL$ source blocks each having length $KL \times T$
bytes and the remaining ZS source blocks each having KS*T bytes.

If Kt*T > F then for encoding purposes, the last symbol MUST be
padded at the end with Kt*T - F zero bytes.

Next, each source block MUST be divided into N = NL + NS contiguous
sub-blocks, the first NL sub-blocks each consisting of K contiguous
sub-symbols of size of TL*Al and the remaining NS sub-blocks each
consisting of K contiguous sub-symbols of size of TS*Al. The symbol
alignment parameter Al ensures that sub-symbols are always a multiple
of A bytes.

Finally, the m-th symbol of a source block consists of the
concatenation of the m-th sub-symbol from each of the N sub-blocks.
Note that this implies that when N > 1 then a symbol is NOT a
contiguous portion of the object.

5.4.2 Encoding packet construction

5.4.2.1 General

Each encoding packet contains the following information:

- Source Block Number (SBN)
- Encoding Symbol ID (ESI)
- encoding symbol(s)

Each source block is encoded independently of the others. Source
blocks are numbered consecutively from zero.

Encoding Symbol ID values from 0 to K-1 identify the source symbols
of a source block in sequential order, where K is the number of
symbols in the source block. Encoding Symbol IDs from K onwards
identify repair symbols.

5.4.2.2 Encoding packet construction

Each encoding packet either consists entirely of source symbols
(source packet) or entirely of repair symbols (repair packet). A
packet may contain any number of symbols from the same source block.
In the case that the last source symbol in a source packet includes
padding bytes added for FEC encoding purposes then these bytes need
not be included in the packet. Otherwise, only whole symbols MUST be
included.

The Encoding Symbol ID, X, carried in each source packet is the
Encoding Symbol ID of the first source symbol carried in that packet. The subsequent source symbols in the packet have Encoding Symbol IDs, X+1 to X+G-1, in sequential order, where G is the number of symbols in the packet.

Similarly, the Encoding Symbol ID, X, placed into a repair packet is the Encoding Symbol ID of the first repair symbol in the repair packet and the subsequent repair symbols in the packet have Encoding Symbol IDs X+1 to X+G-1 in sequential order, where G is the number of symbols in the packet.

Note that it is not necessary for the receiver to know the total number of repair packets.

Associated with each symbol is a triple of integers (d, a, b).

The G repair symbol triples \((d[0], a[0], b[0]), \ldots, (d[G-1], a[G-1], b[G-1])\) for the repair symbols placed into a repair packet with ESI X are computed using the Triple generator defined in Section 5.5.4.4 as follows:

\[
\text{For each } i = 0, \ldots, G-1, \ (d[i], a[i], b[i]) = \text{Trip}[K,X+i]
\]

The G repair symbols to be placed in repair packet with ESI X are calculated based on the repair symbol triples as described in Section 5.5 using the intermediate symbols C and the LT encoder \(\text{LTenc}[K, C, (d[i], a[i], b[i])]\).

5.5 Systematic Raptor encoder

5.5.1 Encoding overview

The systematic Raptor encoder is used to generate repair symbols from a source block that consists of K source symbols.

Symbols are the fundamental data units of the encoding and decoding process. For each source block (sub-block) all symbols (sub-symbols) are the same size. The atomic operation performed on symbols (sub-symbols) for both encoding and decoding is the exclusive-or operation.

Let \(C'[0], \ldots, C'[K-1]\) denote the K source symbols.

Let \(C[0], \ldots, C[L-1]\) denote L intermediate symbols.

The first step of encoding is to generate a number, \(L > K\), of intermediate symbols from the K source symbols. In this step, K source triples \((d[0], a[0], b[0]), \ldots, (d[K-1], a[K-1], b[K-1])\) are
generated using the Trip[] generator as described in Section 5.5.2.2. The K source triples are associated with the K source symbols and are then used to determine the L intermediate symbols C[0], ..., C[L-1] from the source symbols using an inverse encoding process. This process can be can be realized by a Raptor decoding process.

Certain "pre-coding relationships" MUST hold within the L intermediate symbols. Section 5.5.2.3 describes these relationships and how the intermediate symbols are generated from the source symbols.

Once the intermediate symbols have been generated, repair symbols are produced and one or more repair symbols are placed as a group into a single data packet. Each repair symbol group is associated with an Encoding Symbol ID (ESI) and a number, G, of encoding symbols. The ESI is used to generate a triple of three integers, (d, a, b) for each repair symbol, again using the Trip[] generator as described in Section 5.5.2.3. Then, each (d,a,b)-triple is used to generate the corresponding repair symbol from the intermediate symbols using the LTEnc[K, C[0], ..., C[L-1], (d,a,b)] generator described in Section 5.5.4.3.

5.5.2 First encoding step: Intermediate Symbol Generation

5.5.2.1 General

The first encoding step is a pre-coding step to generate the L intermediate symbols C[0], ..., C[L-1] from the source symbols C'[0], ..., C'[K-1]. The intermediate symbols are uniquely defined by two sets of constraints:

1. The intermediate symbols are related to the source symbols by a set of source symbol triples. The generation of the source symbol triples is defined in Section 5.5.2.2 using the the Trip[] generator described in Section 5.5.4.4.

2. A set of pre-coding relationships hold within the intermediate symbols themselves. These are defined in Section 5.5.2.3

The generation of the L intermediate symbols is then defined in Section 5.5.2.4

5.5.2.2 Source symbol triples

Each of the K source symbols is associated with a triple (d[i], a[i], b[i]) for 0 <= i < K. The source symbol triples are determined using the Triple generator defined in Section 5.5.4.4 as:
For each i, 0 <= i < K
(d[i], a[i], b[i]) = Trip[K, i]

5.5.2.3 Pre-coding relationships

The pre-coding relationships amongst the L intermediate symbols are defined by expressing the last L-K intermediate symbols in terms of the first K intermediate symbols.

The last L-K intermediate symbols C[K],...,C[L-1] consist of S LDPC symbols and H Half symbols. The values of S and H are determined from K as described below. Then L = K+S+H.

Let

X be the smallest positive integer such that X*(X-1) >= 2*K.
S be the smallest prime integer such that S >= ceil(0.01*K) + X
H be the smallest integer such that choose(H,ceil(H/2)) >= K + S
H' = ceil(H/2)
L = K+S+H

C[0],...,C[K-1] denote the first K intermediate symbols
C[K],...,C[K+S-1] denote the S LDPC symbols, initialised to zero
C[K+S],...,C[L-1] denote the H Half symbols, initialised to zero

The S LDPC symbols are defined to be the values of C[K],...,C[K+S-1] at the end of the following process:

For i = 0,...,K-1 do

\[ a = 1 + \left(\text{floor}(i/S) \mod (S-1)\right) \]

\[ b = i \mod S \]

\[ C[K + b] = C[K + b] \oplus C[i] \]

\[ b = (b + a) \mod S \]
The H Half symbols are defined as follows:

Let

\[ g[i] = i \oplus \left\lfloor \frac{i}{2} \right\rfloor \]

Note: \( g[i] \) is the Gray sequence, in which each element differs from the previous one in a single bit position.

\( g[j,k] \) denote the jth element, \( j=0, 1, 2, \ldots \), of the subsequence of \( g[i] \) whose elements have exactly \( k \) non-zero bits in their binary representation.

Then, the Half symbols are defined as the values of \( C[K+S], \ldots, C[L-1] \) after the following process:

For \( h = 0, \ldots, H-1 \) do

For \( j = 0, \ldots, K+S-1 \) do

If bit \( h \) of \( g[j,H'] \) is equal to 1 then \( C[h+K+S] = C[h+K+S] \oplus C[j] \).

5.5.2.4 Intermediate symbols

5.5.2.4.1 Definition

Given the K source symbols \( C'[0], C'[1], \ldots, C'[K-1] \) the L intermediate symbols \( C[0], C[1], \ldots, C[L-1] \) are the uniquely defined symbol values that satisfy the following conditions:

1. The K source symbols \( C'[0], C'[1], \ldots, C'[K-1] \) satisfy the K constraints \( C'[i] = \text{LTEnc}[K, (C[0], \ldots, C[L-1]), (d[i], a[i], b[i])] \), for all i, \( 0 \leq i < K \).

2. The L intermediate symbols \( C[0], C[1], \ldots, C[L-1] \) satisfy the pre-coding relationships defined in Section 5.5.2.3
5.5.2.4.2 Example method for calculation of intermediate symbols

This subsection describes a possible method for calculation of the L intermediate symbols $C[0], C[1], ..., C[L-1]$ satisfying the constraints in Section 5.5.2.4.1.

The generator matrix $G$ for a code which generates $N$ output symbols from $K$ input symbols is an $N \times K$ matrix over GF(2), where each row corresponds to one of the output symbols and each column to one of the input symbols and where the $i$th output symbol is equal to the sum of those input symbols whose column contains a non-zero entry in row $i$.

Then, the $L$ intermediate symbols can be calculated as follows:

Let

- $C$ denote the column vector of the $L$ intermediate symbols, $C[0], C[1], ..., C[L-1]$.
- $D$ denote the column vector consisting of $S+H$ zero symbols followed by the $K$ source symbols $C'[0], C'[1], ..., C'[K-1]$.

Then the above constraints define an $L \times L$ matrix over GF(2), $A$, such that:

- $A \cdot C = D$

The matrix $A$ can be constructed as follows:

Let:

- $G_{LDPC}$ be the $S \times K$ generator matrix of the LDPC symbols. So,
  \[ G_{LDPC} \cdot \text{Transpose}([C[0], ..., C[K-1]]) = \text{Transpose}([C[K], ..., C[K+S-1]]) \]
- $G_{Half}$ be the $H \times (K+S)$ generator matrix of the Half symbols, So,
  \[ G_{Half} \cdot \text{Transpose}([C[0], ..., C[S+K-1]]) = \text{Transpose}([C[K+S], ..., C[K+S+H-1]]) \]
- $I_S$ be the $S \times S$ identity matrix
- $I_H$ be the $H \times H$ identity matrix
- $0_{SxH}$ be the $S \times H$ zero matrix
G_LT be the KxL generator matrix of the encoding symbols generated by the LT Encoder. So,

\[
G_LT \times \text{Transpose}[(C[0], \ldots, C[L-1])] = \\
\text{Transpose}[(C'[0], C'[1], \ldots, C'[K-1])]
\]

i.e. \( G_LT(i, j) = 1 \) if and only if \( C[j] \) is included in the symbols which are XORed to produce \( \text{LTEnc}[K, (C[0], \ldots, C[L-1]), (d[i], a[i], b[i])] \).

Then:

The first \( S \) rows of \( A \) are equal to \( G_{LDPC} \mid I_S \mid 0_{SxH} \).

The next \( H \) rows of \( A \) are equal to \( G_{Half} \mid I_H \).

The remaining \( K \) rows of \( A \) are equal to \( G_LT \).

The matrix \( A \) is depicted in Figure 3 below:

\[
\begin{array}{c|c|c}
K & S & H \\
\hline
S & G_{LDPC} & I_S & 0_{SxH} \\
\hline
H & G_{Half} & I_H \\
\hline
K & G_LT \\
\end{array}
\]

Figure 3: The matrix \( A \)

The intermediate symbols can then be calculated as:

\[
C = (A^{-1}) \times D
\]

The source triples are generated such that for any \( K \) matrix \( A \) has full rank and is therefore invertible. This calculation can be realized by applying a Raptor decoding process to the \( K \) source symbols \( C'[0], C'[1], \ldots, C'[K-1] \) to produce the \( L \) intermediate symbols \( C[0], C[1], \ldots, C[L-1] \).
To efficiently generate the intermediate symbols from the source symbols, it is recommended that an efficient decoder implementation such as that described in Section 5.6 be used. The source symbol triples are designed to facilitate efficient decoding of the source symbols using that algorithm.

5.5.3 Second encoding step: LT encoding

In the second encoding step, the repair symbol with ESI X is generated by applying the generator \( \text{LTenc}[K, (C[0], C[1],..., C[L-1]), (d, a, b)] \) defined in Section 5.5.4.3 to the L intermediate symbols \( C[0], C[1],..., C[L-1] \) using the triple \( (d, a, b) = \text{Trip}[K, X] \) generated according to Section 5.4.2.

5.5.4 Generators

5.5.4.1 Random Generator

The random number generator \( \text{Rand}[X, i, m] \) is defined as follows, where \( X \) is a non-negative integer, \( i \) is a non-negative integer and \( m \) is a positive integer and the value produced is an integer between 0 and \( m-1 \). Let \( V0 \) and \( V1 \) be arrays of 256 entries each, where each entry is a 4-byte unsigned integer. These arrays are provided in Section 5.7.

Then,

\[
\text{Rand}[X, i, m] = (V0[(X + i) \mod 256] \oplus V1[(\text{floor}(X/256) + i) \mod 256]) \mod m
\]

5.5.4.2 Degree Generator

The degree generator \( \text{Deg}[v] \) is defined as follows, where \( v \) is an integer that is at least 0 and less than \( 2^{20} = 1048576 \).

In Table 1, find the index \( j \) such that \( f[j-1] <= v < f[j] \)

Then, \( \text{Deg}[v] = d[j] \)
Table 1: Defines the degree distribution for encoding symbols

<table>
<thead>
<tr>
<th>Index j</th>
<th>f[j]</th>
<th>d[j]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>--</td>
</tr>
<tr>
<td>1</td>
<td>10241</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>491582</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>712794</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>831695</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>948446</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>1032189</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>1048576</td>
<td>40</td>
</tr>
</tbody>
</table>

5.5.4.3 LT Encoding Symbol Generator

The encoding symbol generator $\text{LTEnc}[K, (C[0], C[1],..., C[L-1]), (d, a, b)]$ takes the following inputs:

- $K$ is the number of source symbols (or sub-symbols) for the source block (sub-block). Let $L$ be derived from $K$ as described in Section 5.5.2.3, and let $L'$ be the smallest prime integer greater than or equal to $L$.

- $(C[0], C[1],..., C[L-1])$ is the array of $L$ intermediate symbols (sub-symbols) generated as described in Section 5.5.2.4.

- $(d, a, b)$ is a source triple determined using the Triple generator defined in Section 5.5.4.4, whereby:
  - $d$ is an integer denoting an encoding symbol degree
  - $a$ is an integer between 1 and $L'$-1 inclusive
  - $b$ is an integer between 0 and $L'$-1 inclusive

The encoding symbol generator produces a single encoding symbol as output, according to the following algorithm:
While (b >= L) do b = (b + a) % L’

\[ \text{LTEnc}[K, (C[0], C[1], \ldots, C[L-1]), (d, a, b)] = C[b]. \]

For \( j = 1, \ldots, \min(d-1, L-1) \) do

\[ b = (b + a) \% L’ \]

While (b >= L) do b = (b + a) % L’

\[ \text{LTEnc}[K, (C[0], C[1], \ldots, C[L-1]), (d, a, b)] = \text{LTEnc}[K, (C[0], C[1], \ldots, C[L-1]), (d, a, b)] \land C[b] \]

### 5.5.4.4 Triple generator

The triple generator \( \text{Trip}[K,X] \) takes the following inputs:

- **K** - The number of source symbols
- **X** - An encoding symbol ID

Let

\[ L \text{ be determined from } K \text{ as described in Section 5.5.2.3} \]

\[ L’ \text{ be the smallest prime that is greater than or equal to } L \]

\[ Q = 65521, \text{ the largest prime smaller than } 2^{16}. \]

\[ J(K) \text{ be the systematic index associated with } K, \text{ as defined in Section 5.8} \]

The output of the triple generator is a triple, \( (d, a, b) \) determined as follows:

\[ A = (53591 + J(K) \times 997) \% Q \]

\[ B = 10267 \times (J(K)+1) \% Q \]

\[ Y = (B + X \times A) \% Q \]

\[ v = \text{Rand}[Y, 0, 2^{20}] \]

\[ d = \text{Deg}[v] \]

\[ a = 1 + \text{Rand}[Y, 1, L’-1] \]
5.6  Example FEC decoder

5.6.1  General

This section describes an efficient decoding algorithm for the Raptor
codes described in this specification. Note that each received
encoding symbol can be considered as the value of an equation amongst
the intermediate symbols. From these simultaneous equations, and the
known pre-coding relationships amongst the intermediate symbols, any
algorithm for solving simultaneous equations can successfully decode
the intermediate symbols and hence the source symbols. However, the
algorithm chosen has a major effect on the computational efficiency
of the decoding.

5.6.2  Decoding a source block

5.6.2.1  General

It is assumed that the decoder knows the structure of the source
block it is to decode, including the symbol size, \( T \), and the number \( K \)
of symbols in the source block.

From the algorithms described in Section 5.5, the Raptor decoder can
calculate the total number \( L = K+S+H \) of pre-coding symbols and
determine how they were generated from the source block to be
decoded. In this description it is assumed that the received
encoding symbols for the source block to be decoded are passed to the
decoder. Furthermore, for each such encoding symbol it is assumed
that the number and set of intermediate symbols whose exclusive-or is
equal to the encoding symbol is passed to the decoder. In the case
of source symbols, the source symbol triples described in
Section 5.5.2.2 indicate the number and set of intermediate symbols
which sum to give each source symbol.

Let \( N \geq K \) be the number of received encoding symbols for a source
block and let \( M = S+H+N \). The following \( M \) by \( L \) bit matrix \( A \) can be
derived from the information passed to the decoder for the source
block to be decoded. Let \( C \) be the column vector of the \( L \)
intermediate symbols, and let \( D \) be the column vector of \( M \) symbols
with values known to the receiver, where the first \( S+H \) of the \( M \)
symbols are zero-valued symbols that correspond to LDPC and Half
symbols (these are check symbols for the LDPC and Half symbols, and
not the LDPC and Half symbols themselves), and the remaining \( N \) of the
\( M \) symbols are the received encoding symbols for the source block.
Then, \( A \) is the bit matrix that satisfies \( A*C = D \), where here *
denotes matrix multiplication over GF[2]. In particular, \( A[i,j] = 1 \) if the intermediate symbol corresponding to index \( j \) is exclusive-ORed into the LDPC, Half or encoding symbol corresponding to index \( i \) in the encoding, or if index \( i \) corresponds to a LDPC or Half symbol and index \( j \) corresponds to the same LDPC or Half symbol. For all other \( i \) and \( j \), \( A[i,j] = 0 \).

Decoding a source block is equivalent to decoding \( C \) from known \( A \) and \( D \). It is clear that \( C \) can be decoded if and only if the rank of \( A \) over GF[2] is \( L \). Once \( C \) has been decoded, missing source symbols can be obtained by using the source symbol triples to determine the number and set of intermediate symbols which MUST be exclusive-ORed to obtain each missing source symbol.

The first step in decoding \( C \) is to form a decoding schedule. In this step \( A \) is converted, using Gaussian elimination (using row operations and row and column reorderings) and after discarding \( M - L \) rows, into the \( L \) by \( L \) identity matrix. The decoding schedule consists of the sequence of row operations and row and column re-orderings during the Gaussian elimination process, and only depends on \( A \) and not on \( D \). The decoding of \( C \) from \( D \) can take place concurrently with the forming of the decoding schedule, or the decoding can take place afterwards based on the decoding schedule.

The correspondence between the decoding schedule and the decoding of \( C \) is as follows. Let \( c[0] = 0, c[1] = 1, \ldots, c[L-1] = L-1 \) and \( d[0] = 0, d[1] = 1, \ldots, d[M-1] = M-1 \) initially.

- Each time row \( i \) of \( A \) is exclusive-ORed into row \( i' \) in the decoding schedule then in the decoding process symbol \( D[d[i]] \) is exclusive-ORed into symbol \( D[d[i']] \).

- Each time row \( i \) is exchanged with row \( i' \) in the decoding schedule then in the decoding process the value of \( d[i] \) is exchanged with the value of \( d[i'] \).

- Each time column \( j \) is exchanged with column \( j' \) in the decoding schedule then in the decoding process the value of \( c[j] \) is exchanged with the value of \( c[j'] \).

From this correspondence it is clear that the total number of exclusive-ORs of symbols in the decoding of the source block is the number of row operations (not exchanges) in the Gaussian elimination. Since \( A \) is the \( L \) by \( L \) identity matrix after the Gaussian elimination and after discarding the last \( M - L \) rows, it is clear at the end of successful decoding that the \( L \) symbols \( D[d[0]], D[d[1]], \ldots, D[d[L-1]] \) are the values of the \( L \) symbols \( C[c[0]], C[c[1]], \ldots, C[c[L-1]] \).
The order in which Gaussian elimination is performed to form the decoding schedule has no bearing on whether or not the decoding is successful. However, the speed of the decoding depends heavily on the order in which Gaussian elimination is performed. (Furthermore, maintaining a sparse representation of A is crucial, although this is not described here). The remainder of this section describes an order in which Gaussian elimination could be performed that is relatively efficient.

### 5.6.2.2 First Phase

The first phase of the Gaussian elimination the matrix A is conceptually partitioned into submatrices. The submatrix sizes are parameterized by non-negative integers i and u which are initialized to 0. The submatrices of A are:

1. The submatrix I defined by the intersection of the first i rows and first i columns. This is the identity matrix at the end of each step in the phase.

2. The submatrix defined by the intersection of the first i rows and all but the first i columns and last u columns. All entries of this submatrix are zero.

3. The submatrix defined by the intersection of the first i columns and all but the first i rows. All entries of this submatrix are zero.

4. The submatrix U defined by the intersection of all the rows and the last u columns.

5. The submatrix V formed by the intersection of all but the first i columns and the last u columns and all but the first i rows.

Figure 4 illustrates the submatrices of A. At the beginning of the first phase \( V = A \). In each step, a row of A is chosen.
The following graph defined by the structure of $V$ is used in determining which row of $A$ is chosen. The columns that intersect $V$ are the nodes in the graph, and the rows that have exactly 2 ones in $V$ are the edges of the graph that connect the two columns (nodes) in the positions of the two ones. A component in this graph is a maximal set of nodes (columns) and edges (rows) such that there is a path between each pair of nodes/edges in the graph. The size of a component is the number of nodes (columns) in the component.

There are at most $L$ steps in the first phase. The phase ends successfully when $i + u = L$, i.e., when $V$ and the all zeroes submatrix above $V$ have disappeared and $A$ consists of $I$, the all zeroes submatrix below $I$, and $U$. The phase ends unsuccessfully in decoding failure if at some step before $V$ disappears there is no non-zero row in $V$ to choose in that step. In each step, a row of $A$ is chosen as follows:

- If all entries of $V$ are zero then no row is chosen and decoding fails.
- Let $r$ be the minimum integer such that at least one row of $A$ has exactly $r$ ones in $V$.
  - If $r \neq 2$ then choose a row with exactly $r$ ones in $V$ with minimum original degree among all such rows.
  - If $r = 2$ then choose any row with exactly 2 ones in $V$ that is part of a maximum size component in the graph defined by $X$.

After the row is chosen in this step the first row of $A$ that intersects $V$ is exchanged with the chosen row so that the chosen row is the first row that intersects $V$. The columns of $A$ among those that intersect $V$ are reordered so that one of the $r$ ones in the chosen row appears in the first column of $V$ and so that the remaining $r-1$ ones...
appear in the last columns of $V$. Then, the chosen row is exclusive-
ORed into all the other rows of $A$ below the chosen row that have a
one in the first column of $V$. Finally, $i$ is incremented by $l$ and $u$ is
incremented by $r-1$, which completes the step.

5.6.2.3 Second Phase

The submatrix $U$ is further partitioned into the first $i$ rows,
$U_{upper}$, and the remaining $M - i$ rows, $U_{lower}$. Gaussian elimination
is performed in the second phase on $U_{lower}$ to either determine that
its rank is less than $u$ (decoding failure) or to convert it into a
matrix where the first $u$ rows is the identity matrix (success of the
second phase). Call this $u$ by $u$ identity matrix $I_u$. The $M - L$ rows
of $A$ that intersect $U_{lower} - I_u$ are discarded. After this phase $A$
has $L$ rows and $L$ columns.

5.6.2.4 Third Phase

After the second phase the only portion of $A$ which needs to be zeroed
out to finish converting $A$ into the $L$ by $L$ identity matrix is
$U_{upper}$. The number of rows $i$ of the submatrix $U_{upper}$ is generally
much larger than the number of columns $u$ of $U_{upper}$. To zero out
$U_{upper}$ efficiently, the following precomputation matrix $U'$ is
computed based on $I_u$ in the third phase and then $U'$ is used in the
fourth phase to zero out $U_{upper}$. The $u$ rows of $I_u$ are partitioned
into $\lceil u/8 \rceil$ groups of 8 rows each. Then, for each group of 8 rows
all non-zero combinations of the 8 rows are computed, resulting in
$2^{8} - 1 = 255$ rows (this can be done with $2^{8} - 8 - 1 = 247$ exclusive-
ors of rows per group, since the combinations of Hamming weight one
that appear in $I_u$ do not need to be recomputed). Thus, the
resulting precomputation matrix $U'$ has $\lceil u/8 \rceil \times 255$ rows and $u$
columns. Note that $U'$ is not formally a part of matrix $A$, but will
be used in the fourth phase to zero out $U_{upper}$.

5.6.2.5 Fourth Phase

For each of the first $i$ rows of $A$, for each group of 8 columns in the
$U_{upper}$ submatrix of this row, if the set of 8 column entries in
$U_{upper}$ are not all zero then the row of the precomputation matrix $U'$
that matches the pattern in the 8 columns is exclusive-ORed into the
row, thus zeroing out those 8 columns in the row at the cost of
exclusive-oring one row of $U'$ into the row.

After this phase $A$ is the $L$ by $L$ identity matrix and a complete
decoding schedule has been successfully formed. Then, as explained
in Section 5.6.2.1, the corresponding decoding consisting of
exclusive-ORing known encoding symbols can be executed to recover the
intermediate symbols based on the decoding schedule. The triples
associated with all source symbols are computed according to Section 5.5.2.2. The triples for received source symbols are used in the decoding. The triples for missing source symbols are used to determine which intermediate symbols need to be exclusive-ORed to recover the missing source symbols.

5.7 Random Numbers

The two tables V0 and V1 described in Section 5.5.4.1 are given below. Each entry is a 32-bit integer in decimal representation.

5.7.1 The table V0

251291136, 3952231631, 3370958628, 4070167936, 123631495, 3351110283, 3218676425, 2011642291, 774603218, 2402805061, 1004366930, 1843948209, 428891132, 3746331984, 1591258008, 3067016507, 1433388735, 504005498, 2032657933, 3419319784, 2805686246, 3102436986, 3808671154, 2501582075, 3978944421, 649743608, 1974987508, 2651273766, 2357956801, 715807172, 2722736134, 191939188, 3535520147, 3277019569, 1470435941, 3763101702, 3232409631, 122701163, 3920852693, 782246947, 372121310, 2995604341, 2045698755, 2332962102, 4005368743, 218596347, 3415381967, 4207612806, 861117671, 3676575285, 2581671944, 331220480, 681232419, 307306866, 4112503940, 1158111502, 709227802, 2724140433, 4201101115, 4215970289, 4048876515, 3031661061, 1909085522, 510985033, 1361682810, 129243379, 3142379587, 2569842483, 303268270, 1658118006, 932109358, 1982290045, 2983082771, 3007670818, 3448104768, 683749698, 778296777, 1399125101, 1939403708, 1692176003, 3868299200, 1242476658, 593093658, 1878973865, 2526292949, 1591602827, 3986158854, 3964389521, 2695031039, 1942050155, 424618399, 1347204291, 2669179716, 2434425874, 2540801947, 1384069776, 4123580443, 1523670218, 2708475297, 1046771089, 2229796016, 125546612, 4213663089, 1521339547, 3041843489, 420130494, 10677091, 515623176, 3457502702, 2115821274, 2720124766, 3242576090, 854310108, 425973987, 325832382, 1796851292, 2462744411, 1976861690, 1408671665, 1228817808, 3917210003, 263976645, 2593764743, 2471651269, 4291353919, 650792940, 1191583883, 304651335, 2466530435, 2549583082, 969168436, 201204792, 2268075521, 1169345068, 3250240009, 3963499681, 2560755113, 911182396, 760842409, 3569308693, 2687243553, 381854665, 2613828404, 2761078866, 1456668111, 883760091, 3294951678, 1604598575, 1985308198, 1014570543, 2724959607, 3062518035, 3115293053, 138853680, 4160398285, 332241130, 2068983570, 2247491078, 3669524410, 1575146607, 828029864, 3732001371, 3422026452, 3370954177, 4006626915, 543812220, 1243116171, 3928372514, 2791443445, 4081325272, 2280435605, 885616073,
5.7.2. The table V1

| 807385413, | 2043073223, | 3336749796, | 1302105833, | 2278607931, | 541015020, | 1684564270, | 372709334, | 3508252125, | 1768346005, | 1270451292, | 2603029534, | 2049387273, | 3891424859, | 2152948345, | 4114760273, | 915180310, | 3754787998, | 700530826, | 2131559305, | 1308908630, | 224437350, | 4065424007, | 3638665944, | 1679385496, | 3431342226, | 1775956665, | 3068494238, | 1124062773, | 1033448464, | 4053936583, | 4209030573, | 2868464263, | 311689386, | 259047959, | 4057180909, | 1575367248, | 415214153, | 110249784, | 3006865921, | 4293710613, | 3501256572, | 998007483, | 499288295, | 1205710710, | 299719489, | 640417429, | 3044194711, | 486690751, | 2686640734, | 2394526209, | 2521660077, | 49993987, | 384385867, | 4201106668, | 415906198, | 19296814, | 2402488007, | 213711934, | 1744097284, | 579965637, | 2037662632, | 852173610, | 2681403713, | 1047144830, | 2982173936, | 910285038, | 4187576520, | 2589870048, | 899488887, | 3292758024, | 506322719, | 176010738, | 1865471968, | 2619324712, | 564829442, | 1996870325, | 339679593, | 4071072948, | 3618966336, | 2111320126, | 1093955153, | 957987696, | 892010560, | 1854601078, | 1873407527, | 2498546499, | 2694156259, | 1927339882, | 1650555729, | 183933047, | 3061444337, | 2067387204, | 228962564, | 3904019414, | 1595995433, | 1780701372, | 2463145963, | 307281463, | 3237929991, | 3852995239, | 2398693510, | 3754138664, | 522074127, | 146352474, | 4104915256, | 3029415884, | 3545667983, | 332039810, | 976628269, | 3123492423, | 3041418372, | 2258059298, | 2139377204, | 3243642973, | 3226247917, | 3674004636, | 2698992189, | 3453843574, | 1963216666, | 3509855005, | 2358481858, | 747331248, | 1957348676, | 1097574450, | 2435697214, | 3870972145, | 1888833893, | 2914085525, | 4161315584, | 1273113343, | 3269644828, | 3681293816, | 412536684, | 1156034077, | 3823026442, | 1066971017, | 3598330293, | 1979273937, | 2079029895, | 1195045909, | 1071986421, | 2712821515, | 3377754595, |
5.8 Systematic Indices J(K)

For each value of K the systematic index J(K) is designed to have the property that the set of source symbol triples (d[0], a[0], b[0]), ..., (d[L-1], a[L-1], b[L-1]) are such that the L intermediate symbols are uniquely defined, i.e. the matrix A in Section 5.5.2.4.2 has full rank and is therefore invertible.

The following is the list of the systematic indices for values of K between 4 and 8192 inclusive,

18, 14, 61, 46, 14, 22, 20, 40, 48, 1, 29, 40, 43, 46, 18, 8, 20, 2, 61, 26, 13, 29, 36, 19, 58, 5, 58, 0, 54, 56, 24, 14, 5, 67, 39, 31, 25, 29, 24, 19, 14, 56, 49, 49, 63, 30, 4, 39, 2, 1, 20, 19, 61, 4, 54, 70, 25, 52, 9, 26, 55, 69, 27, 68, 75, 19, 64, 57, 45, 3, 37, 31, 100, 41, 25, 41, 53, 23, 9, 31, 26, 30, 30, 46, 90, 50, 13, 90, 77, 61, 31, 54, 54, 3, 21, 66, 21, 11, 23, 11, 29, 21, 7, 1, 27, 4, 34, 17, 85, 69, 17, 75, 93, 57, 0, 53, 71, 88, 119, 88, 90, 22, 0, 58, 41, 22, 96, 26, 79, 118, 19, 3, 81, 72, 50, 0, 32, 79, 28, 25, 12, 25, 29, 3, 37, 30, 30, 41, 84, 32, 31, 61, 32, 61, 7, 56, 54, 39, 33, 66, 29, 3, 14, 75, 75, 78, 84, 75, 84, 25, 54, 25, 25, 107, 78, 27, 73, 0, 49, 96, 53, 50, 21, 10, 73, 58, 65, 27, 3, 27, 28, 15, 54, 45, 69, 29, 3, 65, 31, 71, 76, 56, 54, 76, 54, 13, 5, 18, 142, 17, 3, 37, 114, 41, 25, 56, 0, 23, 3, 41, 22, 22, 31, 18, 48, 31, 58, 37, 75,
6. Security Considerations

The security considerations for this document are the same as they are for [2].
7. IANA Considerations

This document registers the value <tbd> in namespace ietf:rmt:fec: encoding for the Raptor FEC Scheme defined in this document.
8. Intellectual Property

Digital Fountain does have intellectual property rights associated with the technology described in this document, and intends to provide a full IPR statement specific to this document to the IETF that will satisfy the requirements of the IETF.
9. Acknowledgements

Numerous editorial improvements and clarifications were made to this specification during the review process within 3GPP. Thanks are due to the members of 3GPP Technical Specification Group SA, Working Group 4, for these.

10. References


Authors’ Addresses

Michael Luby
Digital Fountain
39141 Civic Center Drive
Suite 300
Fremont, CA  94538
U.S.A.

Email: luby@digitalfountain.com
Amin Shokrollahi  
EPFL  
Laboratory of Algorithmic Mathematics  
IC-IIF-ALGO  
PSE-A  
Lausanne  1015  
Switzerland  

Email: amin.shokrollahi@epfl.ch

Mark Watson  
Digital Fountain  
39141 Civic Center Drive  
Suite 300  
Fremont, CA  94538  
U.S.A.

Email: mark@digitalfountain.com

Thomas Stockhammer  
Siemens AG  
Mobile Devices  
Munich  81667  
Germany

Email: stockhammer@nomor.de
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.