             Low Density Parity Check (LDPC) Staircase and Triangle
                   Forward Error Correction (FEC) Schemes

Status of This Memo

Abstract

   This document describes two Fully-Specified Forward Error Correction
   (FEC) Schemes, Low Density Parity Check (LDPC) Staircase and LDPC
   Triangle, and their application to the reliable delivery of data
   objects on the packet erasure channel (i.e., a communication path
   where packets are either received without any corruption or discarded
   during transmission). These systematic FEC codes belong to the well-
   known class of "Low Density Parity Check" codes, and are large block
   FEC codes in the sense of RFC 3453.

Table of Contents

1. Introduction

   [RFC3453] introduces large block FEC codes as an alternative to small
   block FEC codes like Reed-Solomon. The main advantage of such large
   block codes is the possibility to operate efficiently on source
   blocks with a size of several tens of thousands (or more) of source
   symbols. The present document introduces the Fully-Specified FEC
   Encoding ID 3 that is intended to be used with the LDPC-Staircase FEC
   codes, and the Fully-Specified FEC Encoding ID 4 that is intended to
   be used with the LDPC-Triangle FEC codes [RN04][MK03]. Both schemes
   belong to the broad class of large block codes. For a definition of
   the term Fully-Specified Scheme, see Section 4 of [RFC5052].

   LDPC codes rely on a dedicated matrix, called a "parity check
   matrix", at the encoding and decoding ends. The parity check matrix
   defines relationships (or constraints) between the various encoding
   symbols (i.e., source symbols and repair symbols), which are later
   used by the decoder to reconstruct the original k source symbols if
   some of them are missing. These codes are systematic, in the sense
   that the encoding symbols include the source symbols in addition to
   the repair symbols.

   Since the encoder and decoder must operate on the same parity check
   matrix, information must be communicated between them as part of the
   FEC Object Transmission Information.

   A publicly available reference implementation of these codes is
   available and distributed under a GNU/LGPL (Lesser General Public
   License) [LDPC-codec]. Besides, the code extracts included in this
   document are directly contributed to the IETF process by the authors
   of this document and by Radford M. Neal.

2. Requirements Notation

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in [RFC2119].

3. Definitions, Notations, and Abbreviations

3.1. Definitions

   This document uses the same terms and definitions as those specified
   in [RFC5052]. Additionally, it uses the following definitions:

      Source Symbol: a unit of data used during the encoding process

Encoding Symbol: a unit of data generated by the encoding process

Repair Symbol: an encoding symbol that is not a source symbol

Code Rate: the k/n ratio, i.e., the ratio between the number of
source symbols and the number of encoding symbols. The code rate
belongs to a]0; 1] interval. A code rate close to 1 indicates
that a small number of repair symbols have been produced during
the encoding process

Systematic Code: FEC code in which the source symbols are part of
the encoding symbols

Source Block: a block of k source symbols that are considered
together for the encoding

Encoding Symbol Group: a group of encoding symbols that are sent
together, within the same packet, and whose relationships to the
source object can be derived from a single Encoding Symbol ID

Source Packet: a data packet containing only source symbols

Repair Packet: a data packet containing only repair symbols

Packet Erasure Channel: a communication path where packets are
either dropped (e.g., by a congested router or because the number
of transmission errors exceeds the correction capabilities of the
physical layer codes) or received. When a packet is received, it
is assumed that this packet is not corrupted

3.2. Notations

This document uses the following notations:

L denotes the object transfer length in bytes.

k denotes the source block length in symbols, i.e., the number of
source symbols of a source block.

n denotes the encoding block length, i.e., the number of encoding
symbols generated for a source block.

E denotes the encoding symbol length in bytes.

B denotes the maximum source block length in symbols, i.e., the
maximum number of source symbols per source block.

N denotes the number of source blocks into which the object shall be partitioned.

G denotes the number of encoding symbols per group, i.e., the number of symbols sent in the same packet.

CR denotes the "code rate", i.e., the k/n ratio.

max_n denotes the maximum number of encoding symbols generated for any source block. This is in particular the number of encoding symbols generated for a source block of size B.

H denotes the parity check matrix.

N1 denotes the target number of "1s" per column in the left side of the parity check matrix.

N1m3 denotes the value N1 - 3, where N1 is the target number of "1s" per column in the left side of the parity check matrix.

pmms_rand(m) denotes the pseudo-random number generator defined in Section 5.7 that returns a new random integer in [0; m-1] each time it is called.

3.3. Abbreviations

This document uses the following abbreviations:

ESI: Encoding Symbol ID

FEC OTI: FEC Object Transmission Information

FPI: FEC Payload ID

LDPC: Low Density Parity Check

PRNG: Pseudo-Random Number Generator

4. Formats and Codes

4.1. FEC Payload IDs

   The FEC Payload ID is composed of the Source Block Number and the
   Encoding Symbol ID:

      The Source Block Number (12-bit field) identifies from which
      source block of the object the encoding symbol(s) in the packet
      payload is(are) generated. There is a maximum of 2^{12} blocks per
      object. Source block numbering starts at 0.

      The Encoding Symbol ID (20-bit field) identifies which encoding
      symbol(s) generated from the source block is(are) carried in the
      packet payload. There is a maximum of 2^{20} encoding symbols per
      block. The first k values (0 to k-1) identify source symbols, the
      remaining n-k values (k to n-k-1) identify repair symbols.

   There MUST be exactly one FEC Payload ID per packet. In the case of
   an Encoding Symbol Group, when multiple encoding symbols are sent in
   the same packet, the FEC Payload ID refers to the first symbol of the
   packet. The other symbols can be deduced from the ESI of the first
   symbol thanks to a dedicated function, as explained in Section 5.6

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Source Block Number  |       Encoding Symbol ID (20 bits)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
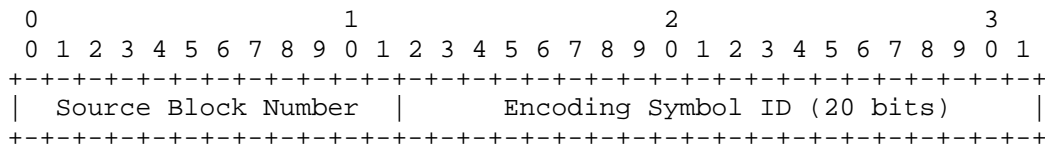
   Figure 1: FEC Payload ID encoding format for FEC Encoding ID 3 and 4

4.2. FEC Object Transmission Information

4.2.1. Mandatory Element

   o FEC Encoding ID: the LDPC-Staircase and LDPC-Triangle Fully-
      Specified FEC Schemes use the FEC Encoding ID 3 (Staircase) and 4
      (Triangle), respectively.

4.2.2. Common Elements

   The following elements MUST be defined with the present FEC Schemes:

   o Transfer-Length (L): a non-negative integer indicating the length
      of the object in bytes. There are some restrictions on the
      maximum Transfer-Length that can be supported:

maximum transfer length = 2^^12 * B * E

For instance, if B=2^^19 (because of a code rate of 1/2,
Section 5.2), and if E=1024 bytes, then the maximum transfer
length is 2^^41 bytes (or 2 TB). The upper limit, with symbols of
size 2^^16-1 bytes and a code rate larger or equal to 1/2, amounts
to 2^^47 bytes (or 128 TB).

o Encoding-Symbol-Length (E): a non-negative integer indicating the
   length of each encoding symbol in bytes.

o Maximum-Source-Block-Length (B): a non-negative integer indicating
   the maximum number of source symbols in a source block. There are
   some restrictions on the maximum B value, as explained in
   Section 5.2.

o Max-Number-of-Encoding-Symbols (max_n): a non-negative integer
   indicating the maximum number of encoding symbols generated for
   any source block. There are some restrictions on the maximum
   max_n value. In particular max_n is at most equal to 2^^20.

Section 5 explains how to define the values of each of these
elements.

4.2.3. Scheme-Specific Elements

The following elements MUST be defined with the present FEC Scheme:

o N1m3: an integer between 0 (default) and 7, inclusive. The target
   number of "1s" per column in the left side of the parity check
   matrix, N1, is then equal to N1m3 + 3 (see Sections 6.2 and 7.2).
   Using the default value of 0 for N1m3 is recommended when the
   sender has no information on the decoding scheme used by the
   receivers. A value greater than 0 for N1m3 can be a good choice
   in specific situations, e.g., with LDPC-staircase codes when the
   sender knows that all the receivers use a Gaussian elimination
   decoding scheme. Nevertheless, the current document does not
   mandate any specific value. This choice is left to the codec
   developer.

o G: an integer between 1 (default) and 31, inclusive, indicating
   the number of encoding symbols per group (i.e., per packet). The
   default value is 1, meaning that each packet contains exactly one
   symbol. Values greater than 1 can also be defined, as explained
   in Section 5.3.

   o PRNG seed: the seed is a 32-bit unsigned integer between 1 and
      0x7FFFFFFE (i.e., 2^^31-2) inclusive. This value is used to
      initialize the Pseudo-Random Number Generator (Section 5.7).

4.2.4. Encoding Format

   This section shows two possible encoding formats of the above FEC
   OTI. The present document does not specify when or how these
   encoding formats should be used.

4.2.4.1. Using the General EXT_FTI Format

   The FEC OTI binary format is the following when the EXT_FTI mechanism
   is used (e.g., within the Asynchronous Layer Coding (ALC)
   [RMT-PI-ALC] or NACK-Oriented Reliable Multicast (NORM) [RMT-PI-NORM]
   protocols).

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   HET = 64    |   HEL = 5     |                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
   |                       Transfer-Length (L)                    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Encoding Symbol Length (E)  | N1m3|    G    |    B (MSB)    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |       B (LSB)        |    Max Nb of Enc. Symbols  (max_n)    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          PRNG seed                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
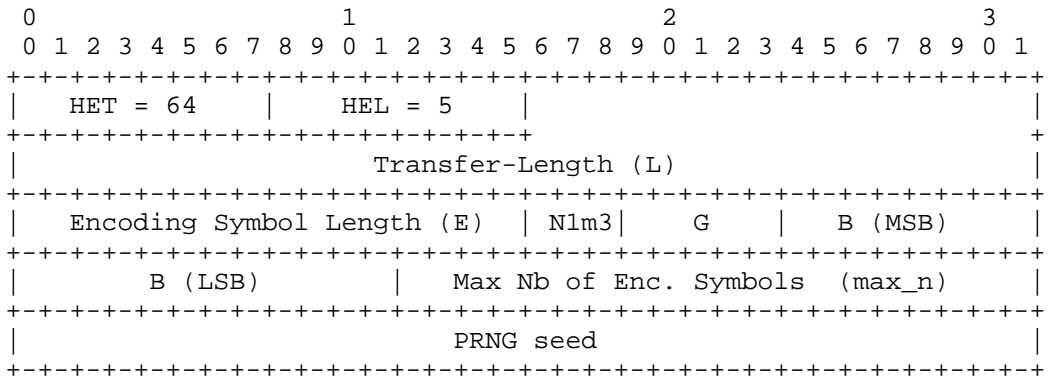
          Figure 2: EXT_FTI Header for FEC Encoding ID 3 and 4

   In particular:

   o The Transfer-Length (L) field size (48 bits) is larger than the
      size required to store the maximum transfer length (Section 4.2.2)
      for field alignment purposes.

   o The Maximum-Source-Block-Length (B) field (20 bits) is split into
      two parts: the 8 most significant bits (MSB) are in the third 32-
      bit word of the EXT_FTI, and the remaining 12 least significant
      bits (LSB) are in the fourth 32-bit word.

4.2.4.2. Using the FDT Instance (FLUTE-Specific)

   When it is desired that the FEC OTI be carried in the File Delivery
   Table (FDT) Instance of a File Delivery over Unidirectional Transport
   (FLUTE) session [RMT-FLUTE], the following XML attributes must be
   described for the associated object:

   o FEC-OTI-FEC-Encoding-ID

   o FEC-OTI-Transfer-length

   o FEC-OTI-Encoding-Symbol-Length

   o FEC-OTI-Maximum-Source-Block-Length

   o FEC-OTI-Max-Number-of-Encoding-Symbols

   o FEC-OTI-Scheme-Specific-Info

   The FEC-OTI-Scheme-Specific-Info contains the string resulting from
   the Base64 encoding [RFC4648] of the following value:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          PRNG seed                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| N1m3|     G     |
+-+-+-+-+-+-+-+-+-+
```
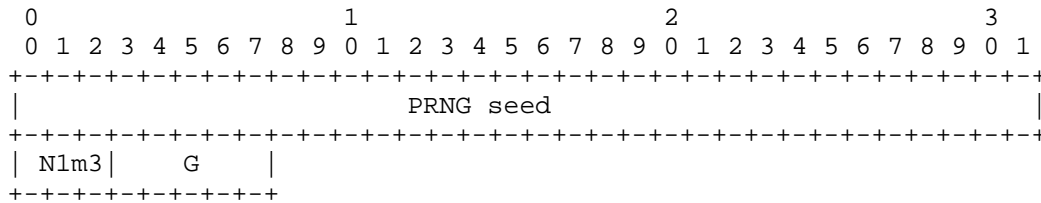
         Figure 3: FEC OTI Scheme-Specific Information to be Included in the
                   FDT Instance for FEC Encoding ID 3 and 4

   During Base64 encoding, the 5 bytes of the FEC OTI Scheme-Specific
   Information are transformed into a string of 8 printable characters
   (in the 64-character alphabet) that is added to the FEC-OTI-Scheme-
   Specific-Info attribute.

5. Procedures

   This section defines procedures that are common to FEC Encoding IDs 3
   and 4.

5.1. General

   The B (maximum source block length in symbols), E (encoding symbol
   length in bytes), and G (number of encoding symbols per group)
   parameters are first determined. The algorithms of Section 5.2 and

Section 5.3 MAY be used to that purpose. Using other algorithms is
possible without compromising interoperability since the B, E, and G
parameters are communicated to the receiver by means of the FEC OTI.

Then, the source object MUST be partitioned using the block
partitioning algorithm specified in [RFC5052]. To that purpose, the
B, L (object transfer length in bytes), and E arguments are provided.
As a result, the object is partitioned into N source blocks. These
blocks are numbered consecutively from 0 to N-1. The first I source
blocks consist of A_large source symbols, the remaining N-I source
blocks consist of A_small source symbols. Each source symbol is E
bytes in length, except perhaps the last symbol, which may be
shorter.

Then, the max_n (maximum number of encoding symbols generated for any
source block) parameter is determined. The algorithm in Section 5.4
MAY be used to that purpose. Using another algorithm is possible
without compromising interoperability since the max_n parameter is
communicated to the receiver by means of the FEC OTI.

For each block, the actual number of encoding symbols, n, MUST then
be determined using the "n-algorithm" detailed in Section 5.5.

Then, FEC encoding and decoding can be done block per block,
independently. To that purpose, a parity check matrix is created,
that forms a system of linear equations between the source and repair
symbols of a given block, where the basic operator is XOR.

This parity check matrix is logically divided into two parts: the
left side (from column 0 to k-1) describes the occurrences of each
source symbol in the system of linear equations; the right side (from
column k to n-1) describes the occurrences of each repair symbol in
the system of linear equations. The only difference between the
LDPC-Staircase and LDPC-Triangle schemes is the construction of this
right sub-matrix. An entry (a "1") in the matrix at position (i,j)
(i.e., at row i and column j) means that the symbol with ESI j
appears in equation i of the system.

When the parity symbols have been created, the sender transmits
source and parity symbols. The way this transmission occurs can
largely impact the erasure recovery capabilities of the LDPC-* FEC.
In particular, sending parity symbols in sequence is suboptimal.
Instead, it is usually recommended to shuffle these symbols. The
interested reader will find more details in [NRFF05].

   The following sections detail how the B, E, G, max_n, and n
   parameters are determined (in Sections 5.2, 5.3, 5.4 and 5.5,
   respectively). Section 5.6 details how Encoding Symbol Groups are
   created, and finally, Section 5.7 covers the PRNG.

5.2. Determining the Maximum Source Block Length (B)

   The B parameter (maximum source block length in symbols) depends on
   several parameters: the code rate (CR), the Encoding Symbol ID field
   length of the FEC Payload ID (20 bits), as well as possible internal
   codec limitations.

   The B parameter cannot be larger than the following values, derived
   from the FEC Payload ID limitations, for a given code rate:

      max1_B = 2^^(20 - ceil(Log2(1/CR)))

   Some common max1_B values are:

   o CR == 1 (no repair symbol): max1_B = 2^^20 = 1,048,576

   o 1/2 <= CR < 1: max1_B = 2^^19 = 524,288 symbols

   o 1/4 <= CR < 1/2: max1_B = 2^^18 = 262,144 symbols

   o 1/8 <= CR < 1/4: max1_B = 2^^17 = 131,072 symbols

   Additionally, a codec MAY impose other limitations on the maximum
   block size. For instance, this is the case when the codec uses
   internally 16-bit unsigned integers to store the Encoding Symbol ID,
   since it does not enable to store all the possible values of a 20-bit
   field. In that case, if for instance, 1/2 <= CR < 1, then the
   maximum source block length is 2^^15. Other limitations may also
   apply, for instance, because of a limited working memory size. This
   decision MUST be clarified at implementation time, when the target
   use case is known. This results in a max2_B limitation.

   Then, B is given by:

      B = min(max1_B, max2_B)

   Note that this calculation is only required at the coder, since the B
   parameter is communicated to the decoder through the FEC OTI.

5.3. Determining the Encoding Symbol Length (E) and Number of Encoding
      Symbols per Group (G)

   The E parameter usually depends on the maximum transmission unit on
   the path (PMTU) from the source to each receiver. In order to
   minimize the protocol header overhead (e.g., the Layered Coding
   Transport (LCT), UDP, IPv4, or IPv6 headers in the case of ALC), E is
   chosen to be as large as possible. In that case, E is chosen so that
   the size of a packet composed of a single symbol (G=1) remains below
   but close to the PMTU.

   However, other considerations can exist. For instance, the E
   parameter can be made a function of the object transfer length.
   Indeed, LDPC codes are known to offer better protection for large
   blocks. In the case of small objects, it can be advantageous to
   reduce the encoding symbol length (E) in order to artificially
   increase the number of symbols and therefore the block size.

   In order to minimize the protocol header overhead, several symbols
   can be grouped in the same Encoding Symbol Group (i.e., G > 1).
   Depending on how many symbols are grouped (G) and on the packet loss
   rate (G symbols are lost for each packet erasure), this strategy
   might or might not be appropriate. A balance must therefore be
   found.

   The current specification does not mandate any value for either E or
   G. The current specification only provides an example of possible
   choices for E and G. Note that this choice is made by the sender,
   and the E and G parameters are then communicated to the receiver
   thanks to the FEC OTI. Note also that the decoding algorithm used
   influences the choice of the E and G parameters. Indeed, increasing
   the number of symbols will negatively impact the processing load when
   decoding is based (in part or totally) on Gaussian elimination,
   whereas the impacts will be rather low when decoding is based on the
   trivial algorithm sketched in Section 6.4.

   Example:

   Let us assume that the trivial decoding algorithm sketched in
   Section 6.4 is used. First, define the target packet payload size,
   pkt_sz (at most equal to the PMTU minus the size of the various
   protocol headers). The pkt_sz must be chosen in such a way that the
   symbol size is an integer. This can require that pkt_sz be a
   multiple of 4, 8, or 16 (see the table below). Then calculate the
   number of packets in the object: nb_pkts = ceil(L / pkt_sz).
   Finally, thanks to nb_pkts, use the following table to find a
   possible G value.

```
+-----------------------+----+------------+-------------------+
|   Number of packets   | G  | Symbol size |         k         |
+-----------------------+----+------------+-------------------+
|      4000 <= nb_pkts  | 1  |   pkt_sz    |     4000 <= k     |
|                       |    |             |                   |
| 1000 <= nb_pkts < 4000| 4  |  pkt_sz / 4 | 4000 <= k < 16000 |
|                       |    |             |                   |
|  500 <= nb_pkts < 1000| 8  |  pkt_sz / 8 | 4000 <= k < 8000  |
|                       |    |             |                   |
|   1 <= nb_pkts < 500  | 16 | pkt_sz / 16 |  16 <= k < 8000   |
+-----------------------+----+------------+-------------------+
```

5.4. Determining the Maximum Number of Encoding Symbols Generated for
      Any Source Block (max_n)

   The following algorithm MAY be used by a sender to determine the
   maximum number of encoding symbols generated for any source block
   (max_n) as a function of B and the target code rate. Since the max_n
   parameter is communicated to the decoder by means of the FEC OTI,
   another method MAY be used to determine max_n.

   Input:

      B: Maximum source block length, for any source block. Section 5.2
      MAY be used to determine its value.

      CR: FEC code rate, which is provided by the user (e.g., when
      starting a FLUTE sending application). It is expressed as a
      floating point value. The CR value must be such that the
      resulting number of encoding symbols per block is at most equal to
      2^{20} (Section 4.1).

   Output:

      max_n: Maximum number of encoding symbols generated for any source
      block.

   Algorithm:

      max_n = ceil(B / CR);

      if (max_n > 2^{20}), then return an error ("invalid code rate");

      (NB: if B has been defined as explained in Section 5.2, this error
      should never happen.)

5.5. Determining the Number of Encoding Symbols of a Block (n)

   The following algorithm, also called "n-algorithm", MUST be used by
   the sender and the receiver to determine the number of encoding
   symbols for a given block (n) as a function of B, k, and max_n.

   Input:

      B: Maximum source block length, for any source block. At a
      sender, Section 5.2 MAY be used to determine its value. At a
      receiver, this value MUST be extracted from the received FEC OTI.

      k: Current source block length. At a sender or receiver, the
      block partitioning algorithm MUST be used to determine its value.

      max_n: Maximum number of encoding symbols generated for any source
      block. At a sender, Section 5.4 MAY be used to determine its
      value. At a receiver, this value MUST be extracted from the
      received FEC OTI.

   Output:

      n: Number of encoding symbols generated for this source block.

   Algorithm:

      n = floor(k * max_n / B);

5.6. Identifying the G Symbols of an Encoding Symbol Group

   When multiple encoding symbols are sent in the same packet, the FEC
   Payload ID information of the packet MUST refer to the first encoding
   symbol. It MUST then be possible to identify each symbol from this
   single FEC Payload ID. To that purpose, the symbols of an Encoding
   Symbol Group (i.e., packet):

   o MUST all be either source symbols or repair symbols. Therefore,
      only source packets and repair packets are permitted, not mixed
      ones.

   o are identified by a function, sender(resp.
      receiver)_find_ESIs_of_group(), that takes as argument:

      * for a sender, the index of the Encoding Symbol Group (i.e.,
         packet) that the application wants to create,

      * for a receiver, the ESI information contained in the FEC
         Payload ID.

      and returns a list of G Encoding Symbol IDs. In the case of a
      source packet, the G Encoding Symbol IDs are chosen consecutively,
      by incrementing the ESI. In the case of a repair packet, the G
      repair symbols are chosen randomly, as explained below.

   o are stored in sequence in the packet, without any padding. In
      other words, the last byte of the i-th symbol is immediately
      followed by the first byte of (i+1)-th symbol.

   The system must first be initialized by creating a random permutation
   of the n-k indexes. This initialization function MUST be called
   immediately after creating the parity check matrix. More precisely,
   since the PRNG seed is not re-initialized, there must not have been a
   call to the PRNG function between the time the parity check matrix
   has been initialized and the time the following initialization
   function is called. This is true both at a sender and at a receiver.

```
   int *txseqToID;
   int *IDtoTxseq;

   /*
    * Initialization function.
    * Warning: use only when G > 1.
    */
   void
   initialize_tables ()
   {
       int i;
       int randInd;
       int backup;

       txseqToID = malloc((n-k) * sizeof(int));
       IDtoTxseq = malloc((n-k) * sizeof(int));
       if (txseqToID == NULL || IDtoTxseq == NULL)
           handle the malloc failures as appropriate...
       /* initialize the two tables that map ID
        * (i.e., ESI-k) to/from TxSequence. */
       for (i = 0; i < n - k; i++) {
           IDtoTxseq[i] = i;
           txseqToID[i] = i;
       }
       /* now randomize everything */
       for (i = 0; i < n - k; i++) {
           randInd = pmms_rand(n - k);
           backup  = IDtoTxseq[i];
           IDtoTxseq[i] = IDtoTxseq[randInd];
           IDtoTxseq[randInd] = backup;
           txseqToID[IDtoTxseq[i]] =  i;
```

```
             txseqToID[IDtoTxseq[randInd]] = randInd;
         }
         return;
     }
```

    It is then possible, at the sender, to determine the sequence of G
    Encoding Symbol IDs that will be part of the group.

```
    /*
     * Determine the sequence of ESIs for the packet under construction
     * at a sender.
     * Warning: use only when G > 1.
     * PktIdx (IN):   index of the packet, in
     *                {0..ceil(k/G)+ceil((n-k)/G)} range
     * ESIs[] (OUT): list of ESIs for the packet
     */
    void
    sender_find_ESIs_of_group (int       PktIdx,
                               ESI_t     ESIs[])
    {
        int i;

        if (PktIdx < nbSourcePkts) {
            /* this is a source packet */
            ESIs[0] = PktIdx * G;
            for (i = 1; i < G; i++) {
                    ESIs[i] = (ESIs[0] + i) % k;
            }
        } else {
            /* this is a repair packet */
            for (i = 0; i < G; i++) {
                ESIs[i] =
                    k +
                    txseqToID[(i + (PktIdx - nbSourcePkts) * G)
                             % (n - k)];
            }
        }
        return;
    }
```

    Similarly, upon receiving an Encoding Symbol Group (i.e., packet), a
    receiver can determine the sequence of G Encoding Symbol IDs from the
    first ESI, esi0, that is contained in the FEC Payload ID.

```
   /*
    * Determine the sequence of ESIs for the packet received.
    * Warning: use only when G > 1.
    * esi0 (IN):  : ESI contained in the FEC Payload ID
    * ESIs[] (OUT): list of ESIs for the packet
    */
   void
   receiver_find_ESIs_of_group (ESI_t    esi0,
                                ESI_t    ESIs[])
   {
       int i;

       if (esi0 < k) {
           /* this is a source packet */
           ESIs[0] = esi0;
           for (i = 1; i < G; i++) {
               ESIs[i] = (esi0 + i) % k;
           }
       } else {
           /* this is a repair packet */
           for (i = 0; i < G; i++) {
               ESIs[i] =
                   k +
                   txseqToID[(i + IDtoTxseq[esi0 - k])
                             % (n - k)];
           }
       }
   }
```

5.7. Pseudo-Random Number Generator

   The FEC Encoding IDs 3 and 4 rely on a pseudo-random number generator
   (PRNG) that must be fully specified, in particular in order to enable
   the receivers and the senders to build the same parity check matrix.

   The Park-Miler "minimal standard" PRNG [PM88] MUST be used. It
   defines a simple multiplicative congruential algorithm: Ij+1 = A * Ij
   (modulo M), with the following choices: A = 7^^5 = 16807 and M =
   2^^31 - 1 = 2147483647. A validation criteria of such a PRNG is the
   following: if seed = 1, then the 10,000th value returned MUST be
   equal to 1043618065.

   Several implementations of this PRNG are known and discussed in the
   literature. An optimized implementation of this algorithm, using
   only 32-bit mathematics, and which does not require any division, can
   be found in [rand31pmc]. It uses the Park and Miller algorithm
   [PM88] with the optimization suggested by D. Carta in [CA90]. The
   history behind this algorithm is detailed in [WI08]. Yet, any other

implementation of the PRNG algorithm that matches the above
validation criteria, like the ones detailed in [PM88], is
appropriate.

This PRNG produces, natively, a 31-bit value between 1 and 0x7FFFFFFE
(2^^31-2) inclusive. Since it is desired to scale the pseudo-random
number between 0 and maxv-1 inclusive, one must keep the most
significant bits of the value returned by the PRNG (the least
significant bits are known to be less random, and modulo-based
solutions should be avoided [PTVF92]). The following algorithm MUST
be used:

Input:

    raw_value: random integer generated by the inner PRNG algorithm,
    between 1 and 0x7FFFFFFE (2^^31-2) inclusive.

    maxv: upper bound used during the scaling operation.

Output:

    scaled_value: random integer between 0 and maxv-1 inclusive.

Algorithm:

    scaled_value = (unsigned long) ((double)maxv * (double)raw_value /
    (double)0x7FFFFFFF);

    (NB: the above C type casting to unsigned long is equivalent to
    using floor() with positive floating point values.)

In this document, pmms_rand(maxv) denotes the PRNG function that
implements the Park-Miller "minimal standard" algorithm, defined
above, and that scales the raw value between 0 and maxv-1 inclusive,
using the above scaling algorithm. Additionally, a function should
be provided to enable the initialization of the PRNG with a seed
(i.e., a 31-bit integer between 1 and 0x7FFFFFFE inclusive) before
calling pmms_rand(maxv) the first time.

6. Full Specification of the LDPC-Staircase Scheme

6.1. General

   The LDPC-Staircase scheme is identified by the Fully-Specified FEC
   Encoding ID 3.

   The PRNG used by the LDPC-Staircase scheme must be initialized by a
   seed. This PRNG seed is an instance-specific FEC OTI attribute
   (Section 4.2.3).

6.2. Parity Check Matrix Creation

   The LDPC-Staircase matrix can be divided into two parts: the left
   side of the matrix defines in which equations the source symbols are
   involved; the right side of the matrix defines in which equations the
   repair symbols are involved.

   The left side MUST be generated by using the following function:

```
/*
 * Initialize the left side of the parity check matrix.
 * This function assumes that an empty matrix of size n-k * k has
 * previously been allocated/reset and that the matrix_has_entry(),
 * matrix_insert_entry() and degree_of_row() functions can access it.
 * (IN): the k, n and N1 parameters.
 */
void left_matrix_init (int k, int n, int N1)
{
    int i;      /* row index or temporary variable */
    int j;      /* column index */
    int h;      /* temporary variable */
    int t;      /* left limit within the list of possible choices u[] */
    int u[N1*MAX_K]; /* table used to have a homogeneous 1 distrib. */

    /* Initialize a list of all possible choices in order to
     * guarantee a homogeneous "1" distribution */
    for (h = N1*k-1; h >= 0; h--) {
        u[h] = h % (n-k);
    }
```

```
    /* Initialize the matrix with N1 "1s" per column, homogeneously */
    t = 0;
    for (j = 0; j < k; j++) { /* for each source symbol column */
        for (h = 0; h < N1; h++) { /* add N1 "1s" */
            /* check that valid available choices remain */
            for (i = t; i < N1*k && matrix_has_entry(u[i], j); i++);
            if (i < N1*k) {
                /* choose one index within the list of possible
                 * choices */
                do {
                    i = t + pmms_rand(N1*k-t);
                } while (matrix_has_entry(u[i], j));
                matrix_insert_entry(u[i], j);

                /* replace with u[t] which has never been chosen */
                u[i] = u[t];
                t++;
            } else {
                /* no choice left, choose one randomly */
                do {
                    i = pmms_rand(n-k);
                } while (matrix_has_entry(i, j));
                matrix_insert_entry(i, j);
            }
        }
    }

    /* Add extra bits to avoid rows with less than two "1s".
     * This is needed when the code rate is smaller than 2/(2+N1) */
    for (i = 0; i < n-k; i++) { /* for each row */
        if (degree_of_row(i) == 0) {
            j = pmms_rand(k);
            matrix_insert_entry(i, j);
        }
        if (degree_of_row(i) == 1) {
            do {
                j = pmms_rand(k);
            } while (matrix_has_entry(i, j));
            matrix_insert_entry(i, j);
        }
    }
}
```

The right side (the staircase) MUST be generated by using the
following function:

```
/*
 * Initialize the right side of the parity check matrix with a
 * staircase structure.
 * (IN): the k and n parameters.
 */
void right_matrix_staircase_init (int k, int n)
{
    int i;        /* row index */

    matrix_insert_entry(0, k);    /* first row */
    for (i = 1; i < n-k; i++) {   /* for the following rows */
        matrix_insert_entry(i, k+i);    /* identity */
        matrix_insert_entry(i, k+i-1); /* staircase */
    }
}
```

Note that just after creating this parity check matrix, when Encoding
Symbol Groups are used (i.e., G > 1), the function initializing the
two random permutation tables (Section 5.6) MUST be called. This is
true both at a sender and at a receiver.

6.3. Encoding

Thanks to the staircase matrix, repair symbol creation is
straightforward: each repair symbol is equal to the sum of all source
symbols in the associated equation, plus the previous repair symbol
(except for the first repair symbol). Therefore, encoding MUST
follow the natural repair symbol order: start with the first repair
symbol and generate a repair symbol with ESI i before a symbol with
ESI i+1.

6.4. Decoding

Decoding basically consists in solving a system of n-k linear
equations whose variables are the n source and repair symbols. Of
course, the final goal is to recover the value of the k source
symbols only.

To that purpose, many techniques are possible. One of them is the
following trivial algorithm [ZP74]: given a set of linear equations,
if one of them has only one remaining unknown variable, then the
value of this variable is that of the constant term. So, replace
this variable by its value in all the remaining linear equations and
reiterate. The value of several variables can therefore be found
recursively. Applied to LDPC FEC codes working over an erasure

Roca, et al.   Standards Track   [Page 21]

channel, the parity check matrix defines a set of linear equations
whose variables are the source symbols and repair symbols. Receiving
or decoding a symbol is equivalent to having the value of a variable.
Appendix A sketches a possible implementation of this algorithm.

A Gaussian elimination (or any optimized derivative) is another
possible decoding technique. Hybrid solutions that start by using
the trivial algorithm above and finish with a Gaussian elimination
are also possible [CR08].

Because interoperability does not depend on the decoding algorithm
used, the current document does not recommend any particular
technique. This choice is left to the codec developer.

However, choosing a decoding technique will have great practical
impacts. It will impact the erasure capabilities: a Gaussian
elimination enables to solve the system with a smaller number of
known symbols compared to the trivial technique. It will also impact
the CPU load: a Gaussian elimination requires more processing than
the above trivial algorithm. Depending on the target use case, the
codec developer will favor one feature or the other.

7. Full Specification of the LDPC-Triangle Scheme

7.1. General

   LDPC-Triangle is identified by the Fully-Specified FEC Encoding ID 4.

   The PRNG used by the LDPC-Triangle scheme must be initialized by a
   seed. This PRNG seed is an instance-specific FEC OTI attribute
   (Section 4.2.3).

7.2. Parity Check Matrix Creation

   The LDPC-Triangle matrix can be divided into two parts: the left side
   of the matrix defines in which equations the source symbols are
   involved; the right side of the matrix defines in which equations the
   repair symbols are involved.

   The left side MUST be generated by using the same left_matrix_init()
   function as with LDPC-Staircase (Section 6.2).

The right side (the triangle) MUST be generated by using the
following function:

```
/*
 * Initialize the right side of the parity check matrix with a
 * triangle structure.
 * (IN): the k and n parameters.
 */
void right_matrix_staircase_init (int k, int n)
{
    int i;        /* row index */
    int j;        /* randomly chosen column indexes in 0..n-k-2 */
    int l;        /* limitation of the # of "1s" added per row */

    matrix_insert_entry(0, k);     /* first row */
    for (i = 1; i < n-k; i++) {    /* for the following rows */
        matrix_insert_entry(i, k+i);    /* identity */
        matrix_insert_entry(i, k+i-1); /* staircase */
        /* now fill the triangle */
        j = i-1;
        for (l = 0; l < j; l++) { /* limit the # of "1s" added */
            j = pmms_rand(j);
            matrix_insert_entry(i, k+j);
        }
    }
}
```

Note that just after creating this parity check matrix, when Encoding
Symbol Groups are used (i.e., G > 1), the function initializing the
two random permutation tables (Section 5.6) MUST be called. This is
true both at a sender and at a receiver.

7.3. Encoding

Here also, repair symbol creation is straightforward: each repair
symbol of ESI i is equal to the sum of all source and repair symbols
(with ESI lower than i) in the associated equation. Therefore,
encoding MUST follow the natural repair symbol order: start with the
first repair symbol, and generate repair symbol with ESI i before
symbol with ESI i+1.

7.4. Decoding

Decoding basically consists in solving a system of n-k linear
equations, whose variables are the n source and repair symbols. Of
course, the final goal is to recover the value of the k source
symbols only. To that purpose, many techniques are possible, as
explained in Section 6.4.

Because interoperability does not depend on the decoding algorithm
used, the current document does not recommend any particular
technique. This choice is left to the codec implementer.

8. Security Considerations

8.1. Problem Statement

A content delivery system is potentially subject to many attacks:
some of them target the network (e.g., to compromise the routing
infrastructure, by compromising the congestion control component),
others target the Content Delivery Protocol (CDP) (e.g., to
compromise its normal behavior), and finally some attacks target the
content itself. Since this document focuses on an FEC building block
independently of any particular CDP (even if ALC and NORM are two
natural candidates), this section only discusses the additional
threats that an arbitrary CDP may be exposed to when using this
building block.

More specifically, several kinds of attacks exist:

o those that are meant to give access to a confidential content
   (e.g., in case of a non-free content),

o those that try to corrupt the object being transmitted (e.g., to
   inject malicious code within an object, or to prevent a receiver
   from using an object), and

o those that try to compromise the receiver's behavior (e.g., by
   making the decoding of an object computationally expensive).

These attacks can be launched either against the data flow itself
(e.g., by sending forged symbols) or against the FEC parameters that
are sent either in-band (e.g., in an EXT_FTI or FDT Instance) or out-
of-band (e.g., in a session description).

8.2. Attacks Against the Data Flow

First of all, let us consider the attacks against the data flow.

8.2.1. Access to Confidential Objects

Access control to a confidential object being transmitted is
typically provided by means of encryption. This encryption can be
done over the whole object (e.g., by the content provider, before the
FEC encoding process), or be done on a packet per packet basis (e.g.,
when IPsec/ESP is used [RFC4303]). If confidentiality is a concern,

   it is RECOMMENDED that one of these solutions be used. Even if we
   mention these attacks here, they are not related or facilitated by
   the use of FEC.

8.2.2. Content Corruption

   Protection against corruptions (e.g., after sending forged packets)
   is achieved by means of a content integrity verification/sender
   authentication scheme. This service can be provided at the object
   level, but in that case a receiver has no way to identify which
   symbol(s) is(are) corrupted if the object is detected as corrupted.
   This service can also be provided at the packet level. In this case,
   after removing all forged packets, the object may be, in some cases,
   recovered. Several techniques can provide this source
   authentication/content integrity service:

   o at the object level, the object MAY be digitally signed (with
      public key cryptography), for instance, by using RSASSA-PKCS1-v1_5
      [RFC3447]. This signature enables a receiver to check the object
      integrity, once the latter has been fully decoded. Even if
      digital signatures are computationally expensive, this calculation
      occurs only once per object, which is usually acceptable;

   o at the packet level, each packet can be digitally signed. A major
      limitation is the high computational and transmission overheads
      that this solution requires (unless perhaps if Elliptic Curve
      Cryptography (ECC) is used). To avoid this problem, the signature
      may span a set of symbols (instead of a single one) in order to
      amortize the signature calculation. But if a single symbol is
      missing, the integrity of the whole set cannot be checked;

   o at the packet level, a Group Message Authentication Code (MAC)
      [RFC2104] scheme can be used, for instance, by using HMAC-SHA-1
      with a secret key shared by all the group members, senders, and
      receivers. This technique creates a cryptographically secured
      (thanks to the secret key) digest of a packet that is sent along
      with the packet. The Group MAC scheme does not create a
      prohibitive processing load or transmission overhead, but it has a
      major limitation: it only provides a group authentication/
      integrity service since all group members share the same secret
      group key, which means that each member can send a forged packet.
      It is therefore restricted to situations where group members are
      fully trusted (or in association with another technique such as a
      pre-check);

   o at the packet level, Timed Efficient Stream Loss-Tolerant
      Authentication (TESLA) [RFC4082] is an attractive solution that is
      robust to losses, provides a true authentication/integrity

service, and does not create any prohibitive processing load or
transmission overhead. Yet, checking a packet requires a small
delay (a second or more) after its reception.

Techniques relying on public key cryptography (digital signatures and
TESLA during the bootstrap process, when used) require that public
keys be securely associated to the entities. This can be achieved by
a Public Key Infrastructure (PKI), or by a PGP Web of Trust, or by
pre-distributing the public keys of each group member.

Techniques relying on symmetric key cryptography (Group MAC) require
that a secret key be shared by all group members. This can be
achieved by means of a group key management protocol, or simply by
pre-distributing the secret key (but this manual solution has many
limitations).

It is up to the CDP developer, who knows the security requirements
and features of the target application area, to define which solution
is the most appropriate. Nonetheless, in case there is any concern
of the threat of object corruption, it is RECOMMENDED that at least
one of these techniques be used.

8.3. Attacks Against the FEC Parameters

Let us now consider attacks against the FEC parameters (or FEC OTI).
The FEC OTI can either be sent in-band (i.e., in an EXT_FTI or in an
FDT Instance containing FEC OTI for the object) or out-of-band (e.g.,
in a session description). Attacks on these FEC parameters can
prevent the decoding of the associated object: for instance,
modifying the B parameter will lead to a different block
partitioning.

It is therefore RECOMMENDED that security measures be taken to
guarantee the FEC OTI integrity. To that purpose, the packets
carrying the FEC parameters sent in-band in an EXT_FTI header
extension SHOULD be protected by one of the per-packet techniques
described above: digital signature, Group MAC, or TESLA. When FEC
OTI is contained in an FDT Instance, this object SHOULD be protected,
for instance, by digitally signing it with XML digital signatures
[RFC3275]. Finally, when FEC OTI is sent out-of-band (e.g., in a
session description) the latter SHOULD be protected, for instance, by
digitally signing it with [RFC3852].

The same considerations concerning the key management aspects apply
here, also.

9. IANA Considerations

   Values of FEC Encoding IDs and FEC Instance IDs are subject to IANA
   registration. For general guidelines on IANA considerations as they
   apply to this document, see [RFC5052].

   This document assigns the Fully-Specified FEC Encoding ID 3 under the
   "ietf:rmt:fec:encoding" name-space to "LDPC Staircase Codes".

   This document assigns the Fully-Specified FEC Encoding ID 4 under the
   "ietf:rmt:fec:encoding" name-space to "LDPC Triangle Codes".

10. Acknowledgments

   Section 5.5 is derived from an earlier document, and we would like to
   thank S. Peltotalo and J. Peltotalo for their contribution. We would
   also like to thank Pascal Moniot, Laurent Fazio, Mathieu Cunche,
   Aurelien Francillon, Shao Wenjian, Magnus Westerlund, Brian
   Carpenter, Tim Polk, Jari Arkko, Chris Newman, Robin Whittle, and
   Alfred Hoenes for their comments.

   Last but not least, the authors are grateful to Radford M. Neal
   (University of Toronto) whose LDPC software
   (http://www.cs.toronto.edu/~radford/ldpc.software.html) inspired this
   work.

11. References

11.1. Normative References

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
                   Requirement Levels", RFC 2119, BCP 14, March 1997.

   [RFC5052]   Watson, M., Luby, M., and L. Vicisano, "Forward Error
                   Correction (FEC) Building Block", RFC 5052,
                   August 2007.

11.2. Informative References

   [ZP74]   Zyablov, V. and M. Pinsker, "Decoding Complexity of
                   Low-Density Codes for Transmission in a Channel with
                   Erasures", Translated from Problemy Peredachi
                   Informatsii, Vol.10, No. 1, pp.15-28, January-
                   March 1974.

   [RN04]   Roca, V. and C. Neumann, "Design, Evaluation and
                     Comparison of Four Large Block FEC Codecs: LDPC, LDGM,
                     LDGM-Staircase and LDGM-Triangle, Plus a Reed-Solomon
                     Small Block FEC Codec", INRIA Research Report RR-5225,
                     June 2004.

   [NRFF05]   Neumann, C., Roca, V., Francillon, A., and D. Furodet,
                     "Impacts of Packet Scheduling and Packet Loss
                     Distribution on FEC Performances: Observations and
                     Recommendations", ACM CoNEXT'05 Conference, Toulouse,
                     France (an extended version is available as INRIA
                     Research Report RR-5578), October 2005.

   [CR08]   Cunche, M. and V. Roca, "Improving the Decoding of
                     LDPC Codes for the Packet Erasure Channel with a
                     Hybrid Zyablov Iterative Decoding/Gaussian Elimination
                     Scheme", INRIA Research Report RR-6473, March 2008.

   [LDPC-codec]   Roca, V., Neumann, C., Cunche, M., and J. Laboure,
                     "LDPC-Staircase/LDPC-Triangle Codec Reference
                     Implementation", INRIA Rhone-Alpes and
                     STMicroelectronics,
                     <http://planete-bcast.inrialpes.fr/>.

   [MK03]   MacKay, D., "Information Theory, Inference and
                     Learning Algorithms", Cambridge University
                     Press, ISBN: 0-521-64298-1, 2003.

   [PM88]   Park, S. and K. Miller, "Random Number Generators:
                     Good Ones are Hard to Find", Communications of the
                     ACM, Vol. 31, No. 10, pp.1192-1201, 1988.

   [CA90]   Carta, D., "Two Fast Implementations of the Minimal
                     Standard Random Number Generator", Communications of
                     the ACM, Vol. 33, No. 1, pp.87-88, January 1990.

   [WI08]   Whittle, R., "Park-Miller-Carta Pseudo-Random Number
                     Generator", January 2008,
                     <http://www.firstpr.com.au/dsp/rand31/>.

   [rand31pmc]   Whittle, R., "31 bit pseudo-random number generator",
                     September 2005, <http://www.firstpr.com.au/dsp/rand31/
                     rand31-park-miller-carta.cc.txt>.

   [PTVF92]   Press, W., Teukolsky, S., Vetterling, W., and B.
                     Flannery, "Numerical Recipes in C; Second Edition",
                     Cambridge University Press, ISBN: 0-521-43108-5, 1992.

   [RMT-PI-ALC]   Luby, M., Watson, M., and L. Vicisano, "Asynchronous
                    Layered Coding (ALC) Protocol Instantiation", Work
                    in Progress, November 2007.

   [RMT-PI-NORM]   Adamson, B., Bormann, C., Handley, M., and J. Macker,
                    "Negative-acknowledgment (NACK)-Oriented Reliable
                    Multicast (NORM) Protocol", Work in Progress,
                    January 2008.

   [RMT-FLUTE]   Paila, T., Walsh, R., Luby, M., Lehtonen, R., and V.
                    Roca, "FLUTE - File Delivery over Unidirectional
                    Transport", Work in Progress, October 2007.

   [RFC3447]   Jonsson, J. and B. Kaliski, "Public-Key Cryptography
                    Standards (PKCS) #1: RSA Cryptography Specifications
                    Version 2.1", RFC 3447, February 2003.

   [RFC4303]   Kent, S., "IP Encapsulating Security Payload (ESP)",
                    RFC 4303, December 2005.

   [RFC2104]   "HMAC: Keyed-Hashing for Message Authentication",
                    RFC 2104, February 1997.

   [RFC4082]   "Timed Efficient Stream Loss-Tolerant Authentication
                    (TESLA): Multicast Source Authentication Transform
                    Introduction", RFC 4082, June 2005.

   [RFC3275]   Eastlake, D., Reagle, J., and D. Solo, "(Extensible
                    Markup Language) XML-Signature Syntax and Processing",
                    RFC 3275, March 2002.

   [RFC3453]   Luby, M., Vicisano, L., Gemmell, J., Rizzo, L.,
                    Handley, M., and J. Crowcroft, "The Use of Forward
                    Error Correction (FEC) in Reliable Multicast",
                    RFC 3453, December 2002.

   [RFC3852]   Housley, R., "Cryptographic Message Syntax (CMS)",
                    RFC 3852, July 2004.

   [RFC4648]   Josefsson, S., "The Base16, Base32, and Base64 Data
                    Encodings", RFC 4648, October 2006.

Appendix A. Trivial Decoding Algorithm (Informative Only)

   A trivial decoding algorithm is sketched below (please see
   [LDPC-codec] for the details omitted here):

   Initialization: allocate a table partial_sum[n-k] of buffers, each
                   buffer being of size the symbol size. There's one
                   entry per equation since the buffers are meant to
                   store the partial sum of each equation; Reset all
                   the buffers to zero;

   /*
    * For each newly received or decoded symbol, try to make progress
    * in the decoding of the associated source block.
    * NB: in case of a symbol group (G>1), this function is called for
    * each symbol of the received packet.
    * NB: a callback function indicates to the caller that new symbol(s)
    *   has(have) been decoded.
    * new_esi (IN): ESI of the new symbol received or decoded
    * new_symb (IN): Buffer of the new symbol received or decoded
    */
   void
   decoding_step(ESI_t   new_esi,
                 symbol_t *new_symb)
   {
       If (new_symb is an already decoded or received symbol) {
           Return;   /* don't waste time with this symbol */
       }

       If (new_symb is the last missing source symbol) {
           Remember that decoding is finished;
           Return;   /* work is over now... */
       }

       Create an empty list of equations having symbols decoded
       during this decoding step;

       /*
        * First add this new symbol to the partial sum of all the
        * equations where the symbol appears.
        */
       For (each equation eq in which new_symb is a variable and
            having more than one unknown variable) {

           Add new_symb to partial_sum[eq];

           Remove entry(eq, new_esi) from the H matrix;

```
        If (the new degree of equation eq == 1) {
            /* a new symbol can be decoded, remember the
             * equation */
            Append eq to the list of equations having symbols
            decoded during this decoding step;
        }
    }

    /*
     * Then finish with recursive calls to decoding_step() for each
     * newly decoded symbol.
     */
    For (each equation eq in the list of equations having symbols
         decoded during this decoding step) {

        /*
         * Because of the recursion below, we need to check that
         * decoding is not finished, and that the equation is
         * __still__ of degree 1
         */
        If (decoding is finished) {
            break;          /* exit from the loop */
        }

        If ((degree of equation eq == 1) {
            Let dec_esi be the ESI of the newly decoded symbol in
            equation eq;

            Remove entry(eq, dec_esi);

            Allocate a buffer, dec_symb, for this symbol and
            copy partial_sum[eq] to dec_symb;

            Inform the caller that a new symbol has been
            decoded via a callback function;

            /* finally, call this function recursively */
            decoding_step(dec_esi, dec_symb);
        }
    }

    Free the list of equations having symbols decoded;
    Return;
}
```

Authors' Addresses

    Vincent Roca
    INRIA
    655, av. de l'Europe
    Inovallee; Montbonnot
    ST ISMIER cedex 38334
    France

    EMail: vincent.roca@inria.fr
    URI: http://planete.inrialpes.fr/people/roca/

    Christoph Neumann
    Thomson
    12, bd de Metz
    Rennes 35700
    France

    EMail: christoph.neumann@thomson.net
    URI: http://planete.inrialpes.fr/people/chneuman/

    David Furodet
    STMicroelectronics
    12, Rue Jules Horowitz
    BP217
    Grenoble Cedex 38019
    France

    EMail: david.furodet@st.com
    URI: http://www.st.com/

Full Copyright Statement

Intellectual Property