



# Intel<sup>®</sup> Intelligent Storage Acceleration Library (Intel<sup>®</sup> ISA-L) Open Source Version

API Reference Manual - Version 2.13

---

*January 30, 2015*

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S LICENSE AGREEMENT FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2011 - 2015 Intel Corporation. All rights reserved.

---

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Storage Library</b>                    | <b>1</b>  |
| 1.1      | About This Document . . . . .             | 1         |
| 1.2      | Overview . . . . .                        | 1         |
| 1.3      | Erasur e Code Functions . . . . .         | 1         |
| 1.4      | System Requirements . . . . .             | 2         |
| <b>2</b> | <b>Function Version Numbers</b>           | <b>3</b>  |
| 2.1      | Function Version Numbers . . . . .        | 3         |
| 2.2      | Function Version Numbers Tables . . . . . | 4         |
| <b>3</b> | <b>Instruction Set Requirements</b>       | <b>6</b>  |
| <b>4</b> | <b>File Index</b>                         | <b>9</b>  |
| 4.1      | File List . . . . .                       | 9         |
| <b>5</b> | <b>File Documentation</b>                 | <b>10</b> |
| 5.1      | erasure_code.h File Reference . . . . .   | 10        |
| 5.1.1    | Detailed Description . . . . .            | 14        |
| 5.1.2    | Function Documentation . . . . .          | 14        |
| 5.1.2.1  | ec_encode_data . . . . .                  | 14        |
| 5.1.2.2  | ec_encode_data_avx . . . . .              | 15        |
| 5.1.2.3  | ec_encode_data_avx2 . . . . .             | 15        |
| 5.1.2.4  | ec_encode_data_base . . . . .             | 15        |
| 5.1.2.5  | ec_encode_data_sse . . . . .              | 15        |
| 5.1.2.6  | ec_encode_data_update . . . . .           | 15        |
| 5.1.2.7  | ec_encode_data_update_avx . . . . .       | 16        |
| 5.1.2.8  | ec_encode_data_update_avx2 . . . . .      | 16        |
| 5.1.2.9  | ec_encode_data_update_base . . . . .      | 16        |
| 5.1.2.10 | ec_encode_data_update_sse . . . . .       | 17        |
| 5.1.2.11 | ec_init_tables . . . . .                  | 17        |
| 5.1.2.12 | gf_2vect_dot_prod_avx . . . . .           | 17        |
| 5.1.2.13 | gf_2vect_dot_prod_avx2 . . . . .          | 18        |
| 5.1.2.14 | gf_2vect_dot_prod_sse . . . . .           | 18        |
| 5.1.2.15 | gf_2vect_mad_avx . . . . .                | 19        |
| 5.1.2.16 | gf_2vect_mad_avx2 . . . . .               | 19        |
| 5.1.2.17 | gf_2vect_mad_sse . . . . .                | 19        |

---

|          |                              |    |
|----------|------------------------------|----|
| 5.1.2.18 | gf_3vect_dot_prod_avx        | 20 |
| 5.1.2.19 | gf_3vect_dot_prod_avx2       | 20 |
| 5.1.2.20 | gf_3vect_dot_prod_sse        | 21 |
| 5.1.2.21 | gf_3vect_mad_avx             | 22 |
| 5.1.2.22 | gf_3vect_mad_avx2            | 22 |
| 5.1.2.23 | gf_3vect_mad_sse             | 22 |
| 5.1.2.24 | gf_4vect_dot_prod_avx        | 23 |
| 5.1.2.25 | gf_4vect_dot_prod_avx2       | 23 |
| 5.1.2.26 | gf_4vect_dot_prod_sse        | 24 |
| 5.1.2.27 | gf_4vect_mad_avx             | 24 |
| 5.1.2.28 | gf_4vect_mad_avx2            | 25 |
| 5.1.2.29 | gf_4vect_mad_sse             | 25 |
| 5.1.2.30 | gf_5vect_dot_prod_avx        | 25 |
| 5.1.2.31 | gf_5vect_dot_prod_avx2       | 26 |
| 5.1.2.32 | gf_5vect_dot_prod_sse        | 26 |
| 5.1.2.33 | gf_5vect_mad_avx             | 27 |
| 5.1.2.34 | gf_5vect_mad_avx2            | 27 |
| 5.1.2.35 | gf_5vect_mad_sse             | 27 |
| 5.1.2.36 | gf_6vect_dot_prod_avx        | 28 |
| 5.1.2.37 | gf_6vect_dot_prod_avx2       | 28 |
| 5.1.2.38 | gf_6vect_dot_prod_sse        | 29 |
| 5.1.2.39 | gf_6vect_mad_avx             | 29 |
| 5.1.2.40 | gf_6vect_mad_avx2            | 29 |
| 5.1.2.41 | gf_6vect_mad_sse             | 30 |
| 5.1.2.42 | gf_gen_cauchy1_matrix        | 30 |
| 5.1.2.43 | gf_gen_rs_matrix             | 30 |
| 5.1.2.44 | gf_inv                       | 31 |
| 5.1.2.45 | gf_invert_matrix             | 31 |
| 5.1.2.46 | gf_mul                       | 31 |
| 5.1.2.47 | gf_vect_dot_prod             | 31 |
| 5.1.2.48 | gf_vect_dot_prod_avx         | 32 |
| 5.1.2.49 | gf_vect_dot_prod_avx2        | 33 |
| 5.1.2.50 | gf_vect_dot_prod_base        | 33 |
| 5.1.2.51 | gf_vect_dot_prod_sse         | 34 |
| 5.1.2.52 | gf_vect_mad                  | 34 |
| 5.1.2.53 | gf_vect_mad_avx              | 35 |
| 5.1.2.54 | gf_vect_mad_avx2             | 35 |
| 5.1.2.55 | gf_vect_mad_base             | 35 |
| 5.1.2.56 | gf_vect_mad_sse              | 35 |
| 5.2      | gf_vect_mul.h File Reference | 35 |
| 5.2.1    | Detailed Description         | 36 |
| 5.2.2    | Function Documentation       | 36 |
| 5.2.2.1  | gf_vect_mul                  | 36 |
| 5.2.2.2  | gf_vect_mul_avx              | 37 |
| 5.2.2.3  | gf_vect_mul_base             | 37 |
| 5.2.2.4  | gf_vect_mul_init             | 37 |

---

---

|              |                        |           |
|--------------|------------------------|-----------|
| 5.2.2.5      | gf_vect_mul_sse        | 38        |
| 5.3          | types.h File Reference | 38        |
| 5.3.1        | Detailed Description   | 38        |
| <b>Index</b> |                        | <b>39</b> |

---

## 1.1 About This Document

This document describes the software programming interface and operation of functions in the library. Sections in this document are grouped by the functions found in individual header files that define the function prototypes. Subsections include function parameters, description and type.

This document refers to the open release version of the library. A separate, general release called the Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) is also available and contains an extended set of functions.

## 1.2 Overview

The Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) Open Source Version is a collection of functions used in storage applications optimized for Intel architecture Intel® 64. In some cases, multiple versions of the same function are available that are optimized for a particular Intel architecture and instruction set. This software takes advantage of new instructions and users should ensure that the chosen function is compatible with hardware it will run on.

## 1.3 Erasure Code Functions

Functions pertaining to erasure codes implement a general Reed-Solomon type encoding for blocks of data to protect against erasure of whole blocks. Individual operations can be described in terms of arithmetic in the Galois finite field  $GF(2^8)$  with the particular field-defining primitive or reducing polynomial  $x^8 + x^4 + x^3 + x^2 + 1$  (0x1d).

For example, the function `ec_encode_data()` will generate a set of parity blocks  $P_i$  from the set of  $k$  source blocks  $D_i$  and arbitrary encoding coefficients  $a_{i,j}$  where each byte in  $P$  is calculated from sources as:

$$P_i = \sum_{j=1}^k a_{i,j} \cdot D_j$$

where addition and multiplication  $\cdot$  is defined in  $GF(2^8)$ . Since any arbitrary set of coefficients  $a_{i,j}$  can be supplied, the same fundamental function can be used for encoding blocks or decoding from blocks in erasure.

As noted in the above document, the scheduler routines do not enforce atomic access to the context structure. If a single scheduler state structure is being used by multiple threads, then the application must take care that calls are not made from different threads at the same time, i.e. thread-safety should be implemented at a level higher than these routines. This could be implemented by employing a separate context structure for each worker thread.

## 1.4 System Requirements

Individual functions may have various run-time requirements such as the minimum version of SSE as described in [Instruction Set Requirements](#). General requirements are listed below.

### Recommended Hardware:

- em64t: A system based on the Intel® Xeon® processor with Intel® 64 architecture.
- IA32: When available for 32-bit functions; A system based on the Intel® Xeon® processor or subsequent IA-32 architecture based processor.

### Software Requirements:

Most functions in the library use the 64-bit embedded and Unix standard for calling convention [http://refspecs.linuxfoundation.org/elf/x86\\_64-abi-0.95.pdf](http://refspecs.linuxfoundation.org/elf/x86_64-abi-0.95.pdf). When available, 32-bit versions use cdecl. Individual functions are written to be statically linked with an application.

### Building Library Functions:

- Yasm Assembler: version at least v1.2.0.

### Building Examples and Tests:

Examples and test source follow simple command line POSIX standards and should be portable to any mostly POSIX-compliant OS.

### Note

Please note that the library assumes 1MB = 1,000,000 bytes in reported performance figures.

---

## CHAPTER 2

# FUNCTION VERSION NUMBERS

---

### 2.1 Function Version Numbers

Individual functions are given version numbers with the format mm-vv-ssss.

- mm = Two hex digits indicating the processor a function was optimized for.

- 00 = Nehalem/Jasper Forest/Multibinary
- 01 = Westmere
- 02 = Sandybridge
- 03 = Ivy Bridge
- 04 = Haswell
- 05 = Silvermont

- vv = function version number

- ssss = function serial number



## 2.2 Function Version Numbers Tables

| Function               | Version    |
|------------------------|------------|
| gf_vect_mul_sse        | 00-02-0034 |
| gf_vect_mul_init       | 00-02-0035 |
| gf_vect_mul_avx        | 01-02-0036 |
| gf_vect_dot_prod_sse   | 00-04-0060 |
| gf_vect_dot_prod_avx   | 02-04-0061 |
| gf_2vect_dot_prod_sse  | 00-03-0062 |
| gf_3vect_dot_prod_sse  | 00-05-0063 |
| gf_4vect_dot_prod_sse  | 00-05-0064 |
| gf_5vect_dot_prod_sse  | 00-04-0065 |
| gf_6vect_dot_prod_sse  | 00-04-0066 |
| ec_init_tables         | 00-01-0068 |
| ec_encode_data_sse     | 00-02-0069 |
| ec_encode_data         | 00-03-0133 |
| gf_vect_mul            | 00-02-0134 |
| ec_encode_data_base    | 00-01-0135 |
| gf_vect_mul_base       | 00-01-0136 |
| gf_vect_dot_prod_base  | 00-01-0137 |
| gf_vect_dot_prod       | 00-02-0138 |
| gf_vect_dot_prod_avx2  | 04-04-0190 |
| gf_2vect_dot_prod_avx  | 02-04-0191 |
| gf_3vect_dot_prod_avx  | 02-04-0192 |
| gf_4vect_dot_prod_avx  | 02-04-0193 |
| gf_5vect_dot_prod_avx  | 02-03-0194 |
| gf_6vect_dot_prod_avx  | 02-03-0195 |
| gf_2vect_dot_prod_avx2 | 04-04-0196 |
| gf_3vect_dot_prod_avx2 | 04-04-0197 |
| gf_4vect_dot_prod_avx2 | 04-04-0198 |
| gf_5vect_dot_prod_avx2 | 04-03-0199 |
| gf_6vect_dot_prod_avx2 | 04-03-019a |
| gf_vect_mad_sse        | 00-00-0200 |
| gf_vect_mad_avx        | 02-00-0201 |
| gf_vect_mad_avx2       | 04-00-0202 |
| gf_2vect_mad_sse       | 00-00-0203 |
| gf_2vect_mad_avx       | 02-00-0204 |
| gf_2vect_mad_avx2      | 04-00-0205 |
| gf_3vect_mad_sse       | 00-00-0206 |
| gf_3vect_mad_avx       | 02-00-0207 |
| gf_3vect_mad_avx2      | 04-00-0208 |
| gf_4vect_mad_sse       | 00-00-0209 |
| gf_4vect_mad_avx       | 02-00-020a |
| gf_4vect_mad_avx2      | 04-00-020b |

---

| Function              | Version    |
|-----------------------|------------|
| gf_5vect_mad_sse      | 00-00-020c |
| gf_5vect_mad_avx      | 02-00-020d |
| gf_5vect_mad_avx2     | 04-00-020e |
| gf_6vect_mad_sse      | 00-00-020f |
| gf_6vect_mad_avx      | 02-00-0210 |
| gf_6vect_mad_avx2     | 04-00-0211 |
| ec_encode_data_update | 00-02-0212 |
| gf_vect_mad           | 00-01-0213 |

---

## CHAPTER 3 INSTRUCTION SET REQUIREMENTS

---

[ec\\_encode\\_data\\_avx](#) (int len, int k, int rows, unsigned char \*gftbbs, unsigned char \*\*data, unsigned char \*\*coding)  
AVX

[ec\\_encode\\_data\\_avx2](#) (int len, int k, int rows, unsigned char \*gftbbs, unsigned char \*\*data, unsigned char \*\*coding)  
AVX2

[ec\\_encode\\_data\\_sse](#) (int len, int k, int rows, unsigned char \*gftbbs, unsigned char \*\*data, unsigned char \*\*coding)  
SSE4.1

[ec\\_encode\\_data\\_update\\_avx](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbbbs, unsigned char \*data, unsigned char \*\*coding)  
AVX

[ec\\_encode\\_data\\_update\\_avx2](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbbbs, unsigned char \*data, unsigned char \*\*coding)  
AVX2

[ec\\_encode\\_data\\_update\\_sse](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbbbs, unsigned char \*data, unsigned char \*\*coding)  
SSE4.1

[gf\\_2vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX

[gf\\_2vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX2

[gf\\_2vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
SSE4.1

[gf\\_2vect\\_mad\\_avx](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX

[gf\\_2vect\\_mad\\_avx2](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX2

[gf\\_2vect\\_mad\\_sse](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
SSE4.1

[gf\\_3vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX

[gf\\_3vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX2

[gf\\_3vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
SSE4.1

---

`gf_3vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX

`gf_3vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX2

`gf_3vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
SSE4.1

`gf_4vect_dot_prod_avx` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX

`gf_4vect_dot_prod_avx2` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX2

`gf_4vect_dot_prod_sse` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
SSE4.1

`gf_4vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX

`gf_4vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX2

`gf_4vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
SSE4.1

`gf_5vect_dot_prod_avx` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX

`gf_5vect_dot_prod_avx2` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX2

`gf_5vect_dot_prod_sse` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
SSE4.1

`gf_5vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX

`gf_5vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
AVX2

`gf_5vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
SSE4.1

`gf_6vect_dot_prod_avx` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX

`gf_6vect_dot_prod_avx2` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
AVX2

`gf_6vect_dot_prod_sse` (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
SSE4.1

---

---

`gf_6vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
AVX

`gf_6vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
AVX2

`gf_6vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
SSE4.1

`gf_vect_dot_prod_avx` (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*dest)  
AVX

`gf_vect_dot_prod_avx2` (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*dest)  
AVX2

`gf_vect_dot_prod_sse` (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*dest)  
SSE4.1

`gf_vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*dest)  
AVX

`gf_vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*dest)  
AVX2

`gf_vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*dest)  
SSE4.1

`gf_vect_mul_avx` (int len, unsigned char \*gftbl, void \*src, void \*dest)  
AVX

`gf_vect_mul_sse` (int len, unsigned char \*gftbl, void \*src, void \*dest)  
SSE4.1

---

## 4.1 File List

Here is a list of all documented files with brief descriptions:

|                                |   |    |
|--------------------------------|---|----|
| <a href="#">erasure_code.h</a> | Interface to functions supporting erasure code encode and decode . . . . .                | 10 |
| <a href="#">gf_vect_mul.h</a>  | Interface to functions for vector (block) multiplication in GF(2 <sup>8</sup> ) . . . . . | 35 |
| <a href="#">types.h</a>        | Defines standard width types . . . . .  | 38 |

## 5.1 erasure\_code.h File Reference

Interface to functions supporting erasure code encode and decode.

```
#include "gf_vect_mul.h"
```

### Functions

- void [ec\\_init\\_tables](#) (int k, int rows, unsigned char \*a, unsigned char \*gftbls)  
*Initialize tables for fast Erasure Code encode and decode.*
- void [ec\\_encode\\_data](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)  
*Generate or decode erasure codes on blocks of data, runs appropriate version.*
- void [ec\\_encode\\_data\\_sse](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)  
*Generate or decode erasure codes on blocks of data.*
- void [ec\\_encode\\_data\\_avx](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)  
*Generate or decode erasure codes on blocks of data.*
- void [ec\\_encode\\_data\\_avx2](#) (int len, int k, int rows, unsigned char \*gftbls, unsigned char \*\*data, unsigned char \*\*coding)  
*Generate or decode erasure codes on blocks of data.*
- void [ec\\_encode\\_data\\_base](#) (int len, int srcs, int dests, unsigned char \*v, unsigned char \*\*src, unsigned char \*\*dest)  
*Generate or decode erasure codes on blocks of data, runs baseline version.*
- void [ec\\_encode\\_data\\_update](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)  
*Generate update for encode or decode of erasure codes from single source, runs appropriate version.*
- void [ec\\_encode\\_data\\_update\\_sse](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)  
*Generate update for encode or decode of erasure codes from single source.*
- void [ec\\_encode\\_data\\_update\\_avx](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)  
*Generate update for encode or decode of erasure codes from single source.*
- void [ec\\_encode\\_data\\_update\\_avx2](#) (int len, int k, int rows, int vec\_i, unsigned char \*g\_tbls, unsigned char \*data, unsigned char \*\*coding)

*Generate update for encode or decode of erasure codes from single source.*

- void [ec\\_encode\\_data\\_update\\_base](#) (int len, int k, int rows, int vec\_i, unsigned char \*v, unsigned char \*data, unsigned char \*\*dest)

*Generate update for encode or decode of erasure codes from single source.*

- void [gf\\_vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*dest)

*GF(2<sup>8</sup>) vector dot product.*

- void [gf\\_vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*dest)

*GF(2<sup>8</sup>) vector dot product.*

- void [gf\\_vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*dest)

*GF(2<sup>8</sup>) vector dot product.*

- void [gf\\_2vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with two outputs.*

- void [gf\\_2vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with two outputs.*

- void [gf\\_2vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with two outputs.*

- void [gf\\_3vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with three outputs.*

- void [gf\\_3vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with three outputs.*

- void [gf\\_3vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with three outputs.*

- void [gf\\_4vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with four outputs.*

- void [gf\\_4vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with four outputs.*

- void [gf\\_4vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with four outputs.*

- void [gf\\_5vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbls, unsigned char \*\*src, unsigned char \*\*dest)

*GF(2<sup>8</sup>) vector dot product with five outputs.*

---



- void [gf\\_5vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with five outputs.*
  - void [gf\\_5vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with five outputs.*
  - void [gf\\_6vect\\_dot\\_prod\\_sse](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with six outputs.*
  - void [gf\\_6vect\\_dot\\_prod\\_avx](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with six outputs.*
  - void [gf\\_6vect\\_dot\\_prod\\_avx2](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector dot product with six outputs.*
  - void [gf\\_vect\\_dot\\_prod\\_base](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector dot product, runs baseline version.*
  - void [gf\\_vect\\_dot\\_prod](#) (int len, int vlen, unsigned char \*gftbbs, unsigned char \*\*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector dot product, runs appropriate version.*
  - void [gf\\_vect\\_mad](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply accumulate, runs appropriate version.*
  - void [gf\\_vect\\_mad\\_sse](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.*
  - void [gf\\_vect\\_mad\\_avx](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.*
  - void [gf\\_vect\\_mad\\_avx2](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.*
  - void [gf\\_vect\\_mad\\_base](#) (int len, int vec, int vec\_i, unsigned char \*v, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply accumulate, baseline version.*
  - void [gf\\_2vect\\_mad\\_sse](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 2 accumulate. SSE version.*
  - void [gf\\_2vect\\_mad\\_avx](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 2 accumulate. AVX version of [gf\\_2vect\\_mad\\_sse\(\)](#).*
  - void [gf\\_2vect\\_mad\\_avx2](#) (int len, int vec, int vec\_i, unsigned char \*gftbbs, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 2 accumulate. AVX2 version of [gf\\_2vect\\_mad\\_sse\(\)](#).*
-

- void `gf_3vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 3 accumulate. SSE version.*
  - void `gf_3vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 3 accumulate. AVX version of `gf_3vect_mad_sse()`.*
  - void `gf_3vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 3 accumulate. AVX2 version of `gf_3vect_mad_sse()`.*
  - void `gf_4vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 4 accumulate. SSE version.*
  - void `gf_4vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 4 accumulate. AVX version of `gf_4vect_mad_sse()`.*
  - void `gf_4vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 4 accumulate. AVX2 version of `gf_4vect_mad_sse()`.*
  - void `gf_5vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 5 accumulate. SSE version.*
  - void `gf_5vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 5 accumulate. AVX version.*
  - void `gf_5vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 5 accumulate. AVX2 version.*
  - void `gf_6vect_mad_sse` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 6 accumulate. SSE version.*
  - void `gf_6vect_mad_avx` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 6 accumulate. AVX version.*
  - void `gf_6vect_mad_avx2` (int len, int vec, int vec\_i, unsigned char \*gftbls, unsigned char \*src, unsigned char \*\*dest)  
*GF(2<sup>8</sup>) vector multiply with 6 accumulate. AVX2 version.*
  - unsigned char `gf_mul` (unsigned char a, unsigned char b)  
*Single element GF(2<sup>8</sup>) multiply.*
  - unsigned char `gf_inv` (unsigned char a)  
*Single element GF(2<sup>8</sup>) inverse.*
  - void `gf_gen_rs_matrix` (unsigned char \*a, int m, int k)  
*Generate a matrix of coefficients to be used for encoding.*
-

- void [gf\\_gen\\_cauchy1\\_matrix](#) (unsigned char \*a, int m, int k)  
*Generate a Cauchy matrix of coefficients to be used for encoding.*
- int [gf\\_invert\\_matrix](#) (unsigned char \*in, unsigned char \*out, const int n)  
*Invert a matrix in  $GF(2^8)$*

### 5.1.1 Detailed Description

Interface to functions supporting erasure code encode and decode. This file defines the interface to optimized functions used in erasure codes. Encode and decode of erasures in  $GF(2^8)$  are made by calculating the dot product of the symbols (bytes in  $GF(2^8)$ ) across a set of buffers and a set of coefficients. Values for the coefficients are determined by the type of erasure code. Using a general dot product means that any sequence of coefficients may be used including erasure codes based on random coefficients. Multiple versions of dot product are supplied to calculate 1-6 output vectors in one pass. Base GF multiply and divide functions can be sped up by defining `GF_LARGE_TABLES` at the expense of memory size.

### 5.1.2 Function Documentation

5.1.2.1 void [ec\\_encode\\_data](#) ( int *len*, int *k*, int *rows*, unsigned char \* *gftbls*, unsigned char \*\* *data*, unsigned char \*\* *coding* )

Generate or decode erasure codes on blocks of data, runs appropriate version.

Given a list of source data blocks, generate one or multiple blocks of encoded data as specified by a matrix of  $GF(2^8)$  coefficients. When given a suitable set of coefficients, this function will perform the fast generation or decoding of Reed-Solomon type erasure codes.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

#### Parameters

|               |   |
|---------------|---|
| <i>len</i>    | Length of each block of data (vector) of source or dest data.   |
| <i>k</i>      | The number of vector sources or rows in the generator matrix for coding.  |
| <i>rows</i>   | The number of output vectors to concurrently encode/decode.   |
| <i>gftbls</i> | Pointer to array of input tables generated from coding coefficients in <a href="#">ec_init_tables()</a> . Must be of size $32*k*rows$ |
| <i>data</i>   | Array of pointers to source input buffers.  |
| <i>coding</i> | Array of pointers to coded output buffers.  |

**Returns**

none

5.1.2.2 `void ec_encode_data_avx ( int len, int k, int rows, unsigned char * gftbls, unsigned char ** data, unsigned char ** coding )`

Generate or decode erasure codes on blocks of data.

Arch specific version of `ec_encode_data()` with same parameters.

**Requires** AVX

5.1.2.3 `void ec_encode_data_avx2 ( int len, int k, int rows, unsigned char * gftbls, unsigned char ** data, unsigned char ** coding )`

Generate or decode erasure codes on blocks of data.

Arch specific version of `ec_encode_data()` with same parameters.

**Requires** AVX2

5.1.2.4 `void ec_encode_data_base ( int len, int srcs, int dests, unsigned char * v, unsigned char ** src, unsigned char ** dest )`

Generate or decode erasure codes on blocks of data, runs baseline version.

Baseline version of `ec_encode_data()` with same parameters.

5.1.2.5 `void ec_encode_data_sse ( int len, int k, int rows, unsigned char * gftbls, unsigned char ** data, unsigned char ** coding )`

Generate or decode erasure codes on blocks of data.

Arch specific version of `ec_encode_data()` with same parameters.

**Requires** SSE4.1

5.1.2.6 `void ec_encode_data_update ( int len, int k, int rows, int vec_i, unsigned char * g_tbls, unsigned char * data, unsigned char ** coding )`

Generate update for encode or decode of erasure codes from single source, runs appropriate version.

---

Given one source data block, update one or multiple blocks of encoded data as specified by a matrix of  $GF(2^8)$  coefficients. When given a suitable set of coefficients, this function will perform the fast generation or decoding of Reed-Solomon type erasure codes from one input source at a time.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

#### Parameters

|               |   |
|---------------|---|
| <i>len</i>    | Length of each block of data (vector) of source or dest data.   |
| <i>k</i>      | The number of vector sources or rows in the generator matrix for coding.  |
| <i>rows</i>   | The number of output vectors to concurrently encode/decode.   |
| <i>vec_i</i>  | The vector index corresponding to the single input source.  |
| <i>g_tbls</i> | Pointer to array of input tables generated from coding coefficients in <a href="#">ec_init_tables()</a> . Must be of size $32*k*rows$ |
| <i>data</i>   | Pointer to single input source used to update output parity.  |
| <i>coding</i> | Array of pointers to coded output buffers.  |

#### Returns

none

**5.1.2.7** `void ec_encode_data_update_avx ( int len, int k, int rows, int vec_i, unsigned char * g_tbls, unsigned char * data, unsigned char ** coding )`

Generate update for encode or decode of erasure codes from single source.

Arch specific version of [ec\\_encode\\_data\\_update\(\)](#) with same parameters.

**Requires** AVX

**5.1.2.8** `void ec_encode_data_update_avx2 ( int len, int k, int rows, int vec_i, unsigned char * g_tbls, unsigned char * data, unsigned char ** coding )`

Generate update for encode or decode of erasure codes from single source.

Arch specific version of [ec\\_encode\\_data\\_update\(\)](#) with same parameters.

**Requires** AVX2

**5.1.2.9** `void ec_encode_data_update_base ( int len, int k, int rows, int vec_i, unsigned char * v, unsigned char * data, unsigned char ** dest )`

Generate update for encode or decode of erasure codes from single source.

Baseline version of [ec\\_encode\\_data\\_update\(\)](#).

---

**5.1.2.10** void `ec_encode_data_update_sse` ( int *len*, int *k*, int *rows*, int *vec\_i*, unsigned char \* *g\_tbls*, unsigned char \* *data*, unsigned char \*\* *coding* )

Generate update for encode or decode of erasure codes from single source.

Arch specific version of `ec_encode_data_update()` with same parameters.

**Requires** SSE4.1

**5.1.2.11** void `ec_init_tables` ( int *k*, int *rows*, unsigned char \* *a*, unsigned char \* *gftbls* )

Initialize tables for fast Erasure Code encode and decode.

Generates the expanded tables needed for fast encode or decode for erasure codes on blocks of data. 32bytes is generated for each input coefficient.

#### Parameters

|               |  |
|---------------|--|
| <i>k</i>      | The number of vector sources or rows in the generator matrix for coding.   |
| <i>rows</i>   | The number of output vectors to concurrently encode/decode.  |
| <i>a</i>      | Pointer to sets of arrays of input coefficients used to encode or decode data.   |
| <i>gftbls</i> | Pointer to start of space for concatenated output tables generated from input coefficients. Must be of size 32*k*rows. |

#### Returns

none

**5.1.2.12** void `gf_2vect_dot_prod_avx` ( int *len*, int *vlen*, unsigned char \* *gftbls*, unsigned char \*\* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector dot product with two outputs.

Vector dot product optimized to calculate two outputs at a time. Does two GF(2<sup>8</sup>) dot products across each byte of the input array and two constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 2\*32\*vlen byte constant array based on the two sets of input coefficients.

**Requires** AVX

#### Parameters

|             |  |
|-------------|--|
| <i>len</i>  | Length of each vector in bytes. Must be >= 16. |
| <i>vlen</i> | Number of vector sources.                      |

|               |   |
|---------------|---|
| <i>gftbls</i> | Pointer to 2*32*vlen byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

**5.1.2.13** `void gf_2vect_dot_prod_avx2 ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with two outputs.

Vector dot product optimized to calculate two outputs at a time. Does two GF(2<sup>8</sup>) dot products across each byte of the input array and two constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 2\*32\*vlen byte constant array based on the two sets of input coefficients.

**Requires** AVX2

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 32$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to 2*32*vlen byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

**5.1.2.14** `void gf_2vect_dot_prod_sse ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with two outputs.

Vector dot product optimized to calculate two outputs at a time. Does two GF(2<sup>8</sup>) dot products across each byte of the input array and two constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 2\*32\*vlen byte constant array based on the two sets of input coefficients.

---

**Requires** SSE4.1

#### Parameters

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 16$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbbs</i> | Pointer to $2 \times 32 \times vlen$ byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

#### Returns

none

**5.1.2.15** `void gf_2vect_mad_avx ( int len, int vec, int vec_i, unsigned char * gftbbs, unsigned char * src, unsigned char ** dest )`

GF( $2^8$ ) vector multiply with 2 accumulate. AVX version of [gf\\_2vect\\_mad\\_sse\(\)](#).

**Requires** AVX

**5.1.2.16** `void gf_2vect_mad_avx2 ( int len, int vec, int vec_i, unsigned char * gftbbs, unsigned char * src, unsigned char ** dest )`

GF( $2^8$ ) vector multiply with 2 accumulate. AVX2 version of [gf\\_2vect\\_mad\\_sse\(\)](#).

**Requires** AVX2

**5.1.2.17** `void gf_2vect_mad_sse ( int len, int vec, int vec_i, unsigned char * gftbbs, unsigned char * src, unsigned char ** dest )`

GF( $2^8$ ) vector multiply with 2 accumulate. SSE version.

Does a GF( $2^8$ ) multiply across each byte of input source with expanded constants and add to destination arrays. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a  $32 \times vec$  byte constant array based on the input coefficients.

**Requires** SSE4.1

#### Parameters

---



|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 32$ .   |
| <i>vec</i>    | The number of vector sources or rows in the generator matrix for coding.  |
| <i>vec_i</i>  | The vector index corresponding to the single input source.  |
| <i>gftbls</i> | Pointer to array of input tables generated from coding coefficients in <code>ec_init_tables()</code> . Must be of size $32 * vec$ . |
| <i>src</i>    | Pointer to source input array.  |
| <i>dest</i>   | Array of pointers to destination input/outputs.   |

**Returns**

none

**5.1.2.18** `void gf_3vect_dot_prod_avx ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char ** dest )`

GF( $2^8$ ) vector dot product with three outputs.

Vector dot product optimized to calculate three outputs at a time. Does three GF( $2^8$ ) dot products across each byte of the input array and three constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a  $3 * 32 * vlen$  byte constant array based on the three sets of input coefficients.

**Requires** AVX

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 16$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to $3 * 32 * vlen$ byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

**5.1.2.19** `void gf_3vect_dot_prod_avx2 ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char ** dest )`

GF( $2^8$ ) vector dot product with three outputs.

---

Vector dot product optimized to calculate three outputs at a time. Does three GF(2<sup>8</sup>) dot products across each byte of the input array and three constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 3\*32\*vlen byte constant array based on the three sets of input coefficients.

**Requires** AVX2

#### Parameters

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 32$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to 3*32*vlen byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

#### Returns

none

**5.1.2.20** `void gf_3vect_dot_prod_sse ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with three outputs.

Vector dot product optimized to calculate three outputs at a time. Does three GF(2<sup>8</sup>) dot products across each byte of the input array and three constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 3\*32\*vlen byte constant array based on the three sets of input coefficients.

**Requires** SSE4.1

#### Parameters

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 16$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to 3*32*vlen byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

5.1.2.21 `void gf_3vect_mad_avx ( int len, int vec, int vec_i, unsigned char * gftbls, unsigned char * src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector multiply with 3 accumulate. AVX version of [gf\\_3vect\\_mad\\_sse\(\)](#).

**Requires** AVX

5.1.2.22 `void gf_3vect_mad_avx2 ( int len, int vec, int vec_i, unsigned char * gftbls, unsigned char * src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector multiply with 3 accumulate. AVX2 version of [gf\\_3vect\\_mad\\_sse\(\)](#).

**Requires** AVX2

5.1.2.23 `void gf_3vect_mad_sse ( int len, int vec, int vec_i, unsigned char * gftbls, unsigned char * src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector multiply with 3 accumulate. SSE version.

Does a GF(2<sup>8</sup>) multiply across each byte of input source with expanded constants and add to destination arrays. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32\*vec byte constant array based on the input coefficients.

**Requires** SSE4.1

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 32$ .   |
| <i>vec</i>    | The number of vector sources or rows in the generator matrix for coding.  |
| <i>vec_i</i>  | The vector index corresponding to the single input source.  |
| <i>gftbls</i> | Pointer to array of input tables generated from coding coefficients in <a href="#">ec_init_tables()</a> . Must be of size 32*vec. |
| <i>src</i>    | Pointer to source input array.  |
| <i>dest</i>   | Array of pointers to destination input/outputs.   |

**Returns**

none

5.1.2.24 `void gf_4vect_dot_prod_avx ( int len, int vlen, unsigned char * gftbbs, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with four outputs.

Vector dot product optimized to calculate four outputs at a time. Does four GF(2<sup>8</sup>) dot products across each byte of the input array and four constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 4\*32\*vlen byte constant array based on the four sets of input coefficients.

**Requires** AVX

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 16$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbbs</i> | Pointer to 4*32*vlen byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

5.1.2.25 `void gf_4vect_dot_prod_avx2 ( int len, int vlen, unsigned char * gftbbs, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with four outputs.

Vector dot product optimized to calculate four outputs at a time. Does four GF(2<sup>8</sup>) dot products across each byte of the input array and four constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 4\*32\*vlen byte constant array based on the four sets of input coefficients.

**Requires** AVX2

---

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 32$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to $4*32*vlen$ byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

**5.1.2.26** `void gf_4vect_dot_prod_sse ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char ** dest )`

GF( $2^8$ ) vector dot product with four outputs.

Vector dot product optimized to calculate four outputs at a time. Does four GF( $2^8$ ) dot products across each byte of the input array and four constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a  $4*32*vlen$  byte constant array based on the four sets of input coefficients.

**Requires** SSE4.1

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 16$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to $4*32*vlen$ byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

**5.1.2.27** `void gf_4vect_mad_avx ( int len, int vec, int vec_i, unsigned char * gftbls, unsigned char * src, unsigned char ** dest )`

GF( $2^8$ ) vector multiply with 4 accumulate. AVX version of `gf_4vect_mad_sse()`.

**Requires** AVX

**5.1.2.28** void `gf_4vect_mad_avx2` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbls*, unsigned char \* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector multiply with 4 accumulate. AVX2 version of `gf_4vect_mad_sse()`.

**Requires** AVX2

**5.1.2.29** void `gf_4vect_mad_sse` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbls*, unsigned char \* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector multiply with 4 accumulate. SSE version.

Does a GF(2<sup>8</sup>) multiply across each byte of input source with expanded constants and add to destination arrays. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32\*vec byte constant array based on the input coefficients.

**Requires** SSE4.1

#### Parameters

|               |  |
|---------------|--|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 32$ .  |
| <i>vec</i>    | The number of vector sources or rows in the generator matrix for coding.   |
| <i>vec_i</i>  | The vector index corresponding to the single input source.   |
| <i>gftbls</i> | Pointer to array of input tables generated from coding coefficients in <code>ec_init_tables()</code> . Must be of size 32*vec. |
| <i>src</i>    | Pointer to source input array.   |
| <i>dest</i>   | Array of pointers to destination input/outputs.  |

#### Returns

none

**5.1.2.30** void `gf_5vect_dot_prod_avx` ( int *len*, int *vlen*, unsigned char \* *gftbls*, unsigned char \*\* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector dot product with five outputs.

Vector dot product optimized to calculate five outputs at a time. Does five GF(2<sup>8</sup>) dot products across each byte of the input array and five constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 5\*32\*vlen byte constant array based on the five sets of input coefficients.

**Requires** AVX

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must $\geq 16$ .  |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to $5*32*vlen$ byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

**5.1.2.31** `void gf_5vect_dot_prod_avx2 ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with five outputs.

Vector dot product optimized to calculate five outputs at a time. Does five GF(2<sup>8</sup>) dot products across each byte of the input array and five constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a  $5*32*vlen$  byte constant array based on the five sets of input coefficients.

**Requires** AVX2

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must $\geq 32$ .  |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to $5*32*vlen$ byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

**5.1.2.32** `void gf_5vect_dot_prod_sse ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with five outputs.

Vector dot product optimized to calculate five outputs at a time. Does five GF(2<sup>8</sup>) dot products across each byte of the input array and five constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding

---

encode and decode. Function requires pre-calculation of a  $5 \times 32 \times vlen$  byte constant array based on the five sets of input coefficients.

**Requires** SSE4.1

#### Parameters

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must $\geq 16$ .  |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to $5 \times 32 \times vlen$ byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

#### Returns

none

**5.1.2.33** `void gf_5vect_mad_avx ( int len, int vec, int vec_i, unsigned char * gftbls, unsigned char * src, unsigned char ** dest )`

GF( $2^8$ ) vector multiply with 5 accumulate. AVX version.

**Requires** AVX

**5.1.2.34** `void gf_5vect_mad_avx2 ( int len, int vec, int vec_i, unsigned char * gftbls, unsigned char * src, unsigned char ** dest )`

GF( $2^8$ ) vector multiply with 5 accumulate. AVX2 version.

**Requires** AVX2

**5.1.2.35** `void gf_5vect_mad_sse ( int len, int vec, int vec_i, unsigned char * gftbls, unsigned char * src, unsigned char ** dest )`

GF( $2^8$ ) vector multiply with 5 accumulate. SSE version.

**Requires** SSE4.1

---



**5.1.2.36** `void gf_6vect_dot_prod_avx ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with six outputs.

Vector dot product optimized to calculate six outputs at a time. Does six GF(2<sup>8</sup>) dot products across each byte of the input array and six constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 6\*32\*vlen byte constant array based on the six sets of input coefficients.

**Requires** AVX

#### Parameters

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 16$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to 6*32*vlen byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

#### Returns

none

**5.1.2.37** `void gf_6vect_dot_prod_avx2 ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char ** dest )`

GF(2<sup>8</sup>) vector dot product with six outputs.

Vector dot product optimized to calculate six outputs at a time. Does six GF(2<sup>8</sup>) dot products across each byte of the input array and six constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 6\*32\*vlen byte constant array based on the six sets of input coefficients.

**Requires** AVX2

#### Parameters

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 32$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to 6*32*vlen byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

**5.1.2.38** void `gf_6vect_dot_prod_sse` ( int *len*, int *vlen*, unsigned char \* *gftbls*, unsigned char \*\* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector dot product with six outputs.

Vector dot product optimized to calculate six outputs at a time. Does six GF(2<sup>8</sup>) dot products across each byte of the input array and six constant sets of coefficients to produce each byte of the outputs. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 6\*32\*vlen byte constant array based on the six sets of input coefficients.

**Requires** SSE4.1

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 16$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to 6*32*vlen byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Array of pointers to destination data buffers.  |

**Returns**

none

**5.1.2.39** void `gf_6vect_mad_avx` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbls*, unsigned char \* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector multiply with 6 accumulate. AVX version.

**Requires** AVX

**5.1.2.40** void `gf_6vect_mad_avx2` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbls*, unsigned char \* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector multiply with 6 accumulate. AVX2 version.

**Requires** AVX2

**5.1.2.41** void `gf_6vect_mad_sse` ( int *len*, int *vec*, int *vec\_i*, unsigned char \* *gftbls*, unsigned char \* *src*, unsigned char \*\* *dest* )

GF(2<sup>8</sup>) vector multiply with 6 accumulate. SSE version.

**Requires** SSE4.1

**5.1.2.42** void `gf_gen_cauchy1_matrix` ( unsigned char \* *a*, int *m*, int *k* )

Generate a Cauchy matrix of coefficients to be used for encoding.

Cauchy matrix example of encoding coefficients where high portion of matrix is identity matrix I and lower portion is constructed as  $1/(i + j) \mid i \neq j, i:\{0,k-1\} j:\{k,m-1\}$ . Any sub-matrix of a Cauchy matrix should be invertable.

#### Parameters

|          |  |
|----------|--|
| <i>a</i> | [mxk] array to hold coefficients                         |
| <i>m</i> | number of rows in matrix corresponding to srcs + parity. |
| <i>k</i> | number of columns in matrix corresponding to srcs.       |

#### Returns

none

**5.1.2.43** void `gf_gen_rs_matrix` ( unsigned char \* *a*, int *m*, int *k* )

Generate a matrix of coefficients to be used for encoding.

Vandermonde matrix example of encoding coefficients where high portion of matrix is identity matrix I and lower portion is constructed as  $2^{i*(j-k+1)} \mid i:\{0,k-1\} j:\{k,m-1\}$ . Commonly used method for choosing coefficients in erasure encoding but does not guarantee invertable for every sub matrix. For large k it is possible to find cases where the decode matrix chosen from sources and parity not in erasure are not invertable. Users may want to adjust for  $k > 5$ .

#### Parameters

|          |  |
|----------|--|
| <i>a</i> | [mxk] array to hold coefficients                         |
| <i>m</i> | number of rows in matrix corresponding to srcs + parity. |
| <i>k</i> | number of columns in matrix corresponding to srcs.       |

#### Returns

none

---

**5.1.2.44 unsigned char gf\_inv ( unsigned char *a* )**

Single element GF(2<sup>8</sup>) inverse.

**Parameters**

|          |               |
|----------|---------------|
| <i>a</i> | Input element |
|----------|---------------|

**Returns**

Field element *b* such that  $a \times b = \{1\}$

**5.1.2.45 int gf\_invert\_matrix ( unsigned char \* *in*, unsigned char \* *out*, const int *n* )**

Invert a matrix in GF(2<sup>8</sup>)

**Parameters**

|            |   |
|------------|---|
| <i>in</i>  | input matrix  |
| <i>out</i> | output matrix such that $[in] \times [out] = [I]$ - identity matrix |
| <i>n</i>   | size of matrix [nxn]  |

**Returns**

0 successful, other fail on singular input matrix

**5.1.2.46 unsigned char gf\_mul ( unsigned char *a*, unsigned char *b* )**

Single element GF(2<sup>8</sup>) multiply.

**Parameters**

|          |                |
|----------|----------------|
| <i>a</i> | Multiplicand a |
| <i>b</i> | Multiplicand b |

**Returns**

Product of *a* and *b* in GF(2<sup>8</sup>)

**5.1.2.47 void gf\_vect\_dot\_prod ( int *len*, int *vlen*, unsigned char \* *gtbIs*, unsigned char \*\* *src*, unsigned char \* *dest* )**

GF(2<sup>8</sup>) vector dot product, runs appropriate version.

---

Does a  $GF(2^8)$  dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a  $32*vlen$  byte constant array based on the input coefficients.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

#### Parameters

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 32$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to $32*vlen$ byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Pointer to destination data array.  |

#### Returns

none

**5.1.2.48** `void gf_vect_dot_prod_avx ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char * dest )`

$GF(2^8)$  vector dot product.

Does a  $GF(2^8)$  dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a  $32*vlen$  byte constant array based on the input coefficients.

**Requires** AVX

#### Parameters

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 16$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to $32*vlen$ byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Pointer to destination data array.  |

**Returns**

none

**5.1.2.49** `void gf_vect_dot_prod_avx2 ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char * dest )`

GF(2<sup>8</sup>) vector dot product.

Does a GF(2<sup>8</sup>) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32\*vlen byte constant array based on the input coefficients.

**Requires** AVX2

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 32$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to 32*vlen byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Pointer to destination data array.  |

**Returns**

none

**5.1.2.50** `void gf_vect_dot_prod_base ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char * dest )`

GF(2<sup>8</sup>) vector dot product, runs baseline version.

Does a GF(2<sup>8</sup>) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32\*vlen byte constant array based on the input coefficients.

**Parameters**

|               |  |
|---------------|--|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 16$ .  |
| <i>vlen</i>   | Number of vector sources.  |
| <i>gftbls</i> | Pointer to 32*vlen byte array of pre-calculated constants based on the array of input coefficients. Only elements 32*CONST*j + 1 of this array are used, where j = (0, 1, 2...) and CONST is the number of elements in the array of input coefficients. The elements used correspond to the original input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.  |
| <i>dest</i>   | Pointer to destination data array.   |

**Returns**

none

5.1.2.51 `void gf_vect_dot_prod_sse ( int len, int vlen, unsigned char * gftbls, unsigned char ** src, unsigned char * dest )`

GF(2<sup>8</sup>) vector dot product.

Does a GF(2<sup>8</sup>) dot product across each byte of the input array and a constant set of coefficients to produce each byte of the output. Can be used for erasure coding encode and decode. Function requires pre-calculation of a 32\*vlen byte constant array based on the input coefficients.

**Requires** SSE4.1

**Parameters**

|               |   |
|---------------|---|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 16$ .   |
| <i>vlen</i>   | Number of vector sources.   |
| <i>gftbls</i> | Pointer to 32*vlen byte array of pre-calculated constants based on the array of input coefficients. |
| <i>src</i>    | Array of pointers to source inputs.   |
| <i>dest</i>   | Pointer to destination data array.  |

**Returns**

none

5.1.2.52 `void gf_vect_mad ( int len, int vec, int vec_i, unsigned char * gftbls, unsigned char * src, unsigned char * dest )`

GF(2<sup>8</sup>) vector multiply accumulate, runs appropriate version.

Does a GF(2<sup>8</sup>) multiply across each byte of input source with expanded constant and add to destination array. Can be used for erasure coding encode and decode update when only one source is available at a time. Function requires pre-calculation of a 32\*vec byte constant array based on the input coefficients.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

**Parameters**

|               |  |
|---------------|--|
| <i>len</i>    | Length of each vector in bytes. Must be $\geq 32$ .  |
| <i>vec</i>    | The number of vector sources or rows in the generator matrix for coding.   |
| <i>vec_i</i>  | The vector index corresponding to the single input source.   |
| <i>gftbls</i> | Pointer to array of input tables generated from coding coefficients in <code>ec_init_tables()</code> . Must be of size 32*vec. |
| <i>src</i>    | Array of pointers to source inputs.  |
| <i>dest</i>   | Pointer to destination data array.   |

**Returns**

none

5.1.2.53 `void gf_vect_mad_avx ( int len, int vec, int vec_i, unsigned char * gftbbs, unsigned char * src, unsigned char * dest )`

GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.

Arch specific version of `gf_vect_mad()` with same parameters.

**Requires** AVX

5.1.2.54 `void gf_vect_mad_avx2 ( int len, int vec, int vec_i, unsigned char * gftbbs, unsigned char * src, unsigned char * dest )`

GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.

Arch specific version of `gf_vect_mad()` with same parameters.

**Requires** AVX2

5.1.2.55 `void gf_vect_mad_base ( int len, int vec, int vec_i, unsigned char * v, unsigned char * src, unsigned char * dest )`

GF(2<sup>8</sup>) vector multiply accumulate, baseline version.

Baseline version of `gf_vect_mad()` with same parameters.

5.1.2.56 `void gf_vect_mad_sse ( int len, int vec, int vec_i, unsigned char * gftbbs, unsigned char * src, unsigned char * dest )`

GF(2<sup>8</sup>) vector multiply accumulate, arch specific version.

Arch specific version of `gf_vect_mad()` with same parameters.

**Requires** SSE4.1

## 5.2 `gf_vect_mul.h` File Reference

Interface to functions for vector (block) multiplication in GF(2<sup>8</sup>).

---



## Functions

- int [gf\\_vect\\_mul\\_sse](#) (int len, unsigned char \*gftbl, void \*src, void \*dest)  
*GF(2<sup>8</sup>) vector multiply by constant.*
- int [gf\\_vect\\_mul\\_avx](#) (int len, unsigned char \*gftbl, void \*src, void \*dest)  
*GF(2<sup>8</sup>) vector multiply by constant.*
- int [gf\\_vect\\_mul](#) (int len, unsigned char \*gftbl, void \*src, void \*dest)  
*GF(2<sup>8</sup>) vector multiply by constant, runs appropriate version.*
- void [gf\\_vect\\_mul\\_init](#) (unsigned char c, unsigned char \*gftbl)  
*Initialize 32-byte constant array for GF(2<sup>8</sup>) vector multiply.*
- void [gf\\_vect\\_mul\\_base](#) (int len, unsigned char \*a, unsigned char \*src, unsigned char \*dest)  
*GF(2<sup>8</sup>) vector multiply by constant, runs baseline version.*

### 5.2.1 Detailed Description

Interface to functions for vector (block) multiplication in GF(2<sup>8</sup>). This file defines the interface to routines used in fast RAID rebuild and erasure codes.

### 5.2.2 Function Documentation

#### 5.2.2.1 int gf\_vect\_mul ( int len, unsigned char \* gftbl, void \* src, void \* dest )

GF(2<sup>8</sup>) vector multiply by constant, runs appropriate version.

Does a GF(2<sup>8</sup>) vector multiply  $b = Ca$  where  $a$  and  $b$  are arrays and  $C$  is a single field element in GF(2<sup>8</sup>). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant  $C$ .  $gftbl(C) = \{C\{00\}, C\{01\}, C\{02\}, \dots, C\{0f\}\}, \{C\{00\}, C\{10\}, C\{20\}, \dots, C\{f0\}\}$ . Len and src must be aligned to 32B.

This function determines what instruction sets are enabled and selects the appropriate version at runtime.

#### Parameters

|              |   |
|--------------|---|
| <i>len</i>   | Length of vector in bytes. Must be aligned to 32B.                  |
| <i>gftbl</i> | Pointer to 32-byte array of pre-calculated constants based on $C$ . |
| <i>src</i>   | Pointer to src data array. Must be aligned to 32B.                  |
| <i>dest</i>  | Pointer to destination data array. Must be aligned to 32B.          |

#### Returns

0 pass, other fail

---

**5.2.2.2 int gf\_vect\_mul\_avx ( int len, unsigned char \* gftbl, void \* src, void \* dest )**

GF(2<sup>8</sup>) vector multiply by constant.

Does a GF(2<sup>8</sup>) vector multiply  $b = Ca$  where  $a$  and  $b$  are arrays and  $C$  is a single field element in GF(2<sup>8</sup>). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant  $C$ .  $gftbl(C) = \{C\{00\}, C\{01\}, C\{02\}, \dots, C\{0f\}\}, \{C\{00\}, C\{10\}, C\{20\}, \dots, C\{f0\}\}$ .  $len$  and  $src$  must be aligned to 32B.

**Requires** AVX

**Parameters**

|              |   |
|--------------|---|
| <i>len</i>   | Length of vector in bytes. Must be aligned to 32B.                  |
| <i>gftbl</i> | Pointer to 32-byte array of pre-calculated constants based on $C$ . |
| <i>src</i>   | Pointer to src data array. Must be aligned to 32B.                  |
| <i>dest</i>  | Pointer to destination data array. Must be aligned to 32B.          |

**Returns**

0 pass, other fail

**5.2.2.3 void gf\_vect\_mul\_base ( int len, unsigned char \* a, unsigned char \* src, unsigned char \* dest )**

GF(2<sup>8</sup>) vector multiply by constant, runs baseline version.

Does a GF(2<sup>8</sup>) vector multiply  $b = Ca$  where  $a$  and  $b$  are arrays and  $C$  is a single field element in GF(2<sup>8</sup>). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant  $C$ .  $gftbl(C) = \{C\{00\}, C\{01\}, C\{02\}, \dots, C\{0f\}\}, \{C\{00\}, C\{10\}, C\{20\}, \dots, C\{f0\}\}$ .  $len$  and  $src$  must be aligned to 32B.

**Parameters**

|             |   |
|-------------|---|
| <i>len</i>  | Length of vector in bytes. Must be aligned to 32B.  |
| <i>a</i>    | Pointer to 32-byte array of pre-calculated constants based on $C$ . only use 2nd element is used. |
| <i>src</i>  | Pointer to src data array. Must be aligned to 32B.  |
| <i>dest</i> | Pointer to destination data array. Must be aligned to 32B.  |

**5.2.2.4 void gf\_vect\_mul\_init ( unsigned char c, unsigned char \* gftbl )**

Initialize 32-byte constant array for GF(2<sup>8</sup>) vector multiply.

Calculates array  $\{C\{00\}, C\{01\}, C\{02\}, \dots, C\{0f\}\}, \{C\{00\}, C\{10\}, C\{20\}, \dots, C\{f0\}\}$  as required by other fast vector multiply functions.

---

**Parameters**

|              |                 |
|--------------|-----------------|
| <i>c</i>     | Constant input. |
| <i>gftbl</i> | Table output.   |

**5.2.2.5 int gf\_vect\_mul\_sse ( int *len*, unsigned char \* *gftbl*, void \* *src*, void \* *dest* )**

GF(2<sup>8</sup>) vector multiply by constant.

Does a GF(2<sup>8</sup>) vector multiply  $b = Ca$  where  $a$  and  $b$  are arrays and  $C$  is a single field element in GF(2<sup>8</sup>). Can be used for RAID6 rebuild and partial write functions. Function requires pre-calculation of a 32-element constant array based on constant  $C$ .  $gftbl(C) = \{C\{00\}, C\{01\}, C\{02\}, \dots, C\{0f\}\}, \{C\{00\}, C\{10\}, C\{20\}, \dots, C\{f0\}\}$ .  $len$  and  $src$  must be aligned to 32B.

**Requires** SSE4.1

**Parameters**

|              |   |
|--------------|---|
| <i>len</i>   | Length of vector in bytes. Must be aligned to 32B.                  |
| <i>gftbl</i> | Pointer to 32-byte array of pre-calculated constants based on $C$ . |
| <i>src</i>   | Pointer to $src$ data array. Must be aligned to 32B.                |
| <i>dest</i>  | Pointer to destination data array. Must be aligned to 32B.          |

**Returns**

0 pass, other fail

**5.3 types.h File Reference**

Defines standard width types.

**5.3.1 Detailed Description**

Defines standard width types.

---

---

ec\_encode\_data  
     erasure\_code.h, 14  
 ec\_encode\_data\_avx  
     erasure\_code.h, 15  
 ec\_encode\_data\_avx2  
     erasure\_code.h, 15  
 ec\_encode\_data\_base  
     erasure\_code.h, 15  
 ec\_encode\_data\_sse  
     erasure\_code.h, 15  
 ec\_encode\_data\_update  
     erasure\_code.h, 15  
 ec\_encode\_data\_update\_avx  
     erasure\_code.h, 16  
 ec\_encode\_data\_update\_avx2  
     erasure\_code.h, 16  
 ec\_encode\_data\_update\_base  
     erasure\_code.h, 16  
 ec\_encode\_data\_update\_sse  
     erasure\_code.h, 16  
 ec\_init\_tables  
     erasure\_code.h, 17  
 erasure\_code.h, 10  
     ec\_encode\_data, 14  
     ec\_encode\_data\_avx, 15  
     ec\_encode\_data\_avx2, 15  
     ec\_encode\_data\_base, 15  
     ec\_encode\_data\_sse, 15  
     ec\_encode\_data\_update, 15  
     ec\_encode\_data\_update\_avx, 16  
     ec\_encode\_data\_update\_avx2, 16  
     ec\_encode\_data\_update\_base, 16  
     ec\_encode\_data\_update\_sse, 16  
     ec\_init\_tables, 17  
     gf\_2vect\_dot\_prod\_avx, 17  
     gf\_2vect\_dot\_prod\_avx2, 18  
     gf\_2vect\_dot\_prod\_sse, 18  
     gf\_2vect\_mad\_avx, 19  
     gf\_2vect\_mad\_avx2, 19  
     gf\_2vect\_mad\_sse, 19  
     gf\_3vect\_dot\_prod\_avx, 20  
     gf\_3vect\_dot\_prod\_avx2, 20  
     gf\_3vect\_dot\_prod\_sse, 21  
     gf\_3vect\_mad\_avx, 22  
     gf\_3vect\_mad\_avx2, 22  
     gf\_3vect\_mad\_sse, 22  
     gf\_4vect\_dot\_prod\_avx, 23  
     gf\_4vect\_dot\_prod\_avx2, 23  
     gf\_4vect\_dot\_prod\_sse, 24  
     gf\_4vect\_mad\_avx, 24  
     gf\_4vect\_mad\_avx2, 24  
     gf\_4vect\_mad\_sse, 25  
     gf\_5vect\_dot\_prod\_avx, 25  
     gf\_5vect\_dot\_prod\_avx2, 26  
     gf\_5vect\_dot\_prod\_sse, 26  
     gf\_5vect\_mad\_avx, 27  
     gf\_5vect\_mad\_avx2, 27  
     gf\_5vect\_mad\_sse, 27  
     gf\_6vect\_dot\_prod\_avx, 27  
     gf\_6vect\_dot\_prod\_avx2, 28  
     gf\_6vect\_dot\_prod\_sse, 29  
     gf\_6vect\_mad\_avx, 29  
     gf\_6vect\_mad\_avx2, 29  
     gf\_6vect\_mad\_sse, 29  
     gf\_gen\_cauchy1\_matrix, 30  
     gf\_gen\_rs\_matrix, 30  
     gf\_inv, 30  
     gf\_invert\_matrix, 31  
     gf\_mul, 31  
     gf\_vect\_dot\_prod, 31  
     gf\_vect\_dot\_prod\_avx, 32  
     gf\_vect\_dot\_prod\_avx2, 33  
     gf\_vect\_dot\_prod\_base, 33  
     gf\_vect\_dot\_prod\_sse, 34  
     gf\_vect\_mad, 34  
     gf\_vect\_mad\_avx, 35  
     gf\_vect\_mad\_avx2, 35  
     gf\_vect\_mad\_base, 35  
     gf\_vect\_mad\_sse, 35  
 gf\_2vect\_dot\_prod\_avx  
     erasure\_code.h, 17  
 gf\_2vect\_dot\_prod\_avx2  
     erasure\_code.h, 18  
 gf\_2vect\_dot\_prod\_sse  
     erasure\_code.h, 18  
 gf\_2vect\_mad\_avx  
     erasure\_code.h, 19

---

gf\_2vect\_mad\_avx2  
  erasure\_code.h, 19

gf\_2vect\_mad\_sse  
  erasure\_code.h, 19

gf\_3vect\_dot\_prod\_avx  
  erasure\_code.h, 20

gf\_3vect\_dot\_prod\_avx2  
  erasure\_code.h, 20

gf\_3vect\_dot\_prod\_sse  
  erasure\_code.h, 21

gf\_3vect\_mad\_avx  
  erasure\_code.h, 22

gf\_3vect\_mad\_avx2  
  erasure\_code.h, 22

gf\_3vect\_mad\_sse  
  erasure\_code.h, 22

gf\_4vect\_dot\_prod\_avx  
  erasure\_code.h, 23

gf\_4vect\_dot\_prod\_avx2  
  erasure\_code.h, 23

gf\_4vect\_dot\_prod\_sse  
  erasure\_code.h, 24

gf\_4vect\_mad\_avx  
  erasure\_code.h, 24

gf\_4vect\_mad\_avx2  
  erasure\_code.h, 24

gf\_4vect\_mad\_sse  
  erasure\_code.h, 25

gf\_5vect\_dot\_prod\_avx  
  erasure\_code.h, 25

gf\_5vect\_dot\_prod\_avx2  
  erasure\_code.h, 26

gf\_5vect\_dot\_prod\_sse  
  erasure\_code.h, 26

gf\_5vect\_mad\_avx  
  erasure\_code.h, 27

gf\_5vect\_mad\_avx2  
  erasure\_code.h, 27

gf\_5vect\_mad\_sse  
  erasure\_code.h, 27

gf\_6vect\_dot\_prod\_avx  
  erasure\_code.h, 27

gf\_6vect\_dot\_prod\_avx2  
  erasure\_code.h, 28

gf\_6vect\_dot\_prod\_sse  
  erasure\_code.h, 29

gf\_6vect\_mad\_avx  
  erasure\_code.h, 29

gf\_6vect\_mad\_avx2  
  erasure\_code.h, 29

gf\_6vect\_mad\_sse  
  erasure\_code.h, 29

gf\_gen\_cauchy1\_matrix  
  erasure\_code.h, 30

gf\_gen\_rs\_matrix  
  erasure\_code.h, 30

gf\_inv  
  erasure\_code.h, 30

gf\_invert\_matrix  
  erasure\_code.h, 31

gf\_mul  
  erasure\_code.h, 31

gf\_vect\_dot\_prod  
  erasure\_code.h, 31

gf\_vect\_dot\_prod\_avx  
  erasure\_code.h, 32

gf\_vect\_dot\_prod\_avx2  
  erasure\_code.h, 33

gf\_vect\_dot\_prod\_base  
  erasure\_code.h, 33

gf\_vect\_dot\_prod\_sse  
  erasure\_code.h, 34

gf\_vect\_mad  
  erasure\_code.h, 34

gf\_vect\_mad\_avx  
  erasure\_code.h, 35

gf\_vect\_mad\_avx2  
  erasure\_code.h, 35

gf\_vect\_mad\_base  
  erasure\_code.h, 35

gf\_vect\_mad\_sse  
  erasure\_code.h, 35

gf\_vect\_mul  
  gf\_vect\_mul.h, 36

gf\_vect\_mul.h, 35

  gf\_vect\_mul, 36

  gf\_vect\_mul\_avx, 36

  gf\_vect\_mul\_base, 37

  gf\_vect\_mul\_init, 37

  gf\_vect\_mul\_sse, 38

gf\_vect\_mul\_avx  
  gf\_vect\_mul.h, 36

---

gf\_vect\_mul\_base  
    gf\_vect\_mul.h, [37](#)  
gf\_vect\_mul\_init  
    gf\_vect\_mul.h, [37](#)  
gf\_vect\_mul\_sse  
    gf\_vect\_mul.h, [38](#)  
  
types.h, [38](#)

---