CUDA-Image-Encryption

Generated by Doxygen 1.8.11

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 common Namespace Reference

Functions

- static void flattenImage (cv::Mat image, uint8_t *&img_vec, uint32_t channels)

  Converts an image of dimensions N x M into a 1D vector of length N x M.
- static void printImageContents (cv::Mat image, uint32_t channels)

  Prints the gray level values in a cv::Mat image in row major order.
- static uint8_t checkOverflow (uint16_t number_1, uint16_t number_2)

  Checks if the product of 2 16-bit unsigned integers exceeds 255.
- static void show_ieee754 (double f)

  formatted output of ieee-754 representation of double-precision floating-point
- static void print_int_bits (int num)

  Print bits of a 32-bit signed integer.
- static uint16_t get_n_mantissa_bits_safe (double f, int number_of_bits)

  Transfers the last n bits from a double to an n-bit unsigned integer.
- static void writeVectorToFile32 (uint32_t *&vec, int length, std::string filename)

  Writes a 32-bit vector to a .txt file.
- static void writeVectorToFile8 (uint8_t *&vec, int length, std::string filename)

  Writes an 8-bit unsigned integer vector to a .txt file.
- static void printArray8 (uint8_t *&arr, int length)

  Prints an 8-bit unsigned integer array of length 'length'.
- static void printArray16 (uint16_t *&arr, int length)

  Prints a 16-bit unsigned integer array of length 'length'.
- static void printArray32 (uint32_t *&arr, int length)

  Prints a 32-bit unsigned integer array of length 'length'.
- static void printArrayDouble (double *&arr, int length)

  Prints a double array of length 'length'.
- static uint32_t getRandomUnsignedInteger32 (uint32_t lower_bound, uint32_t upper_bound)

  Returns a random 32-bit unsigned integer within a range of (lower_bound,upper_bound).
- static int getRandomInteger (int lower_bound, int upper_bound)

  Returns a random 32-bit signed integer within a range of (lower_bound,upper_bound).
- static uint8_t getRandomUnsignedInteger8 (uint8_t lower_bound, uint8_t upper_bound)

  Returns a random 8-bit unsigned integer within a range of (lower_bound,upper_bound).
- static double getRandomDouble (double lower_limit, double upper_limit)

Returns a random double within a range of (lower_bound,upper_bound).

- static config::ChaoticMap mapAssigner (int lower_limit, int upper_limit)

  Returns a value of type ChaoticMap within a range of (lower_limit,upper_limit).
- static void rowColLUTGen (uint32_t *&rowSwapLUT, uint32_t *&rowRandVec, uint32_t *&colSwapLUT, uint32_t *&colRandVec, uint32_t m, uint32_t n)

  Generates shuffled row and column Lookup Tables using Fisher - Yates Shuffle for row and column rotation or swapping.
- static void swapLUT (uint32_t *&swapLUT, uint32_t *randVec, uint32_t m)

  Shuffles the Lookup Table used to shuffle chaotic map choices array.
- static void genLUTVec (uint32_t *&lut_vec, uint32_t n)

  Generates a Lookup Table with values from 0 to n - 1 in ascending order for row and column swapping or rotating.
- static void genMapLUTVec (uint32_t *&lut_vec, uint32_t n)

  Generates a Lookup Table with values from 1 to n in ascending order for shuffling the chaotic map choices array.
- static std::string getFileNameFromPath (std::string filename)

  Gets the file name from the given file path.
- static std::string sha256_hash_string (unsigned char hash[SHA256_DIGEST_LENGTH])

  Converts an 8-bit unsigned char SHA256 array into a SHA256 std::string.
- static std::string calc_sha256 (const char *path)

  Calculates SHA256 Hash of a given file.
- static void checkImageVectors (uint8_t *plain_img_vec, uint8_t *decrypted_img_vec, uint32_t total)

  Finds differences between two image vectors.
- static bool checkImages (cv::Mat image_1, cv::Mat image_2)

  Finds differences between two 2D N x M images.

4.1.1 Function Documentation

4.1.1.1 static void common::flattenImage (cv::Mat *image*, uint8_t *& *img_vec*, uint32_t *channels*) [inline], [static]

Converts an image of dimensions N x M into a 1D vector of length N x M.

Takes a 2D N X M image, a 1D vector of length N X M, and the number of channels as arguments

Definition at line 78 of file commonheader.hpp.

Referenced by main().

```
79   {
80
81     uint16_t m=0,n=0;
82     uint32_t total=0;
83     m=(uint16_t)image.rows;
84     n=(uint16_t)image.cols;
85     total=m*n;
86     image=image.reshape(1,1);
87     for(int i=0;i<total * channels;++i)
88     {
89       img_vec[i]=image.at<uint8_t>(i);
90     }
91   }
```

Here is the caller graph for this function:

4.1.1.2 static void common::printImageContents (cv::Mat *image*, uint32_t *channels*) [inline],[static]

Prints the gray level values in a cv::Mat image in row major order.

Takes 2D N X M image and number of channels as parameters

Definition at line 96 of file commonheader.hpp.

```
97   {
98     for(uint32_t i=0;i<image.rows;++i)
99     {
100       printf("\n");
101       for(uint32_t j=0;j<image.cols;++j)
102       {
103          for(uint32_t k=0;k < channels;++k)
104          {
105
106           printf("%d\t",image.at<Vec3b>(i,j)[k]);
107          }
108        }
109     }
110   }
```

4.1.1.3 static uint8_t common::checkOverflow (uint16_t *number_1*, uint16_t *number_2*) [inline],[static]

Checks if the product of 2 16-bit unsigned integers exceeds 255.

Takes the 2 16-bit unsigned integers as arguments

Definition at line 115 of file commonheader.hpp.

```
116   {
117
118     if((number_1*number_2)>=512)
119     {
120       printf("\n%d , %d exceeded 512",number_1,number_2);
121       return 2;
122     }
123
124     if((number_1*number_2)>=256)
125     {
126       printf("\n%d , %d exceeded 255",number_1,number_2);
127       return 1;
128     }
129     return 0;
130   }
```

4.1.1.4 static void common::show_ieee754 (double *f*) [inline],[static]

formatted output of ieee-754 representation of double-precision floating-point

Definition at line 135 of file commonheader.hpp.

References BIT_RETURN.

```
136   {
137     union {
138         double f;
139         uint32_t u;
140     } fu = { .f = f };
141     int i = sizeof f * CHAR_BIT;
142
143     printf ("  ");
144     while (i--)
145         printf ("%d ", BIT_RETURN(fu.u,i));
146
147     putchar ('\n');
148     printf (" |- - - - - - - - - - - - - - - - - - - - - - - - "
149             "- - - - - - - - -|\n");
150     printf (" |s|      exp      |                 mantissa"
151             "                  |\n\n");
152   }
```

4.1.1.5 static void common::print_int_bits (int *num*) [inline], [static]

Print bits of a 32-bit signed integer.

Takes the number of bits as argument

Definition at line 158 of file commonheader.hpp.

```
159   {
160     int x=1;
161     for(int bit=(sizeof(int)*8)-1; bit>=0;bit--)
162     {
163       /*printf("%i ", num & 0x01);
164       num = num >> 1;*/
165       printf("%c",(num & (x << bit)) ? '1' : '0');
166     }
167   }
```

4.1.1.6 static uint16_t common::get_n_mantissa_bits_safe (double *f,* int *number_of_bits*) [inline], [static]

Transfers the last n bits from a double to an n-bit unsigned integer.

Takes the double and and the number of bits as arguments

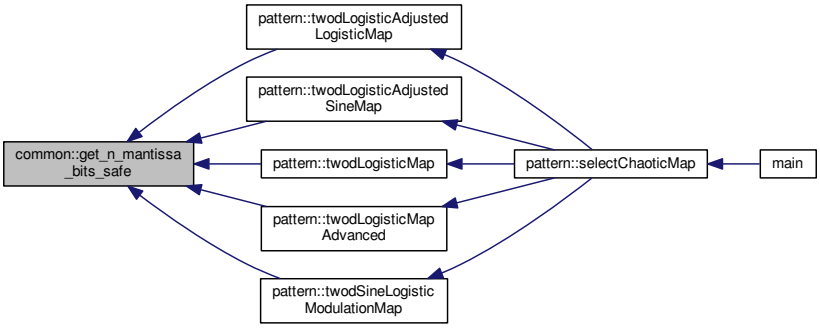
Definition at line 172 of file commonheader.hpp.

References BIT_RETURN.

Referenced by pattern::twodLogisticAdjustedLogisticMap(), pattern::twodLogisticAdjustedSineMap(), pattern::twodLogisticMap(), pattern::twodLogisticMapAdvanced(), and pattern::twodSineLogisticModulationMap().

```
173   {
174     union {
175         double f;
176         uint32_t u;
177     } fu = { .f = f };
178
179     int i=number_of_bits;
180     int bit_store_32=0;
181     uint8_t bit_store_8=0;
182     uint16_t bit_store_16=0;
183
184     while (i--)
185     {
186
187         if(BIT_RETURN(fu.u,i)==1)
188         {
189             bit_store_16 |= 1 << i;
190         }
191
192     }
193
194     return bit_store_16;
195   }
```

Here is the caller graph for this function:



4.1.1.7 static void common::writeVectorToFile32 (uint32_t *& *vec,* int *length,* std::string *filename*) [inline], [static]

Writes a 32-bit vector to a .txt file.

Takes a vector of length 'length', and its length as arguments

Definition at line 200 of file commonheader.hpp.

```
201   {
202     std::ofstream file(filename);
203     if(!file)
204     {
205       cout<<"\nCould not create "<<filename<<"\nExiting...";
206       exit(0);
207     }
208
209     std::string elements = std::string("");
210
211     for(int i = 0; i < length; ++i)
212     {
213       elements.append(std::to_string(vec[i]));
214       elements.append("\n");
215     }
216     file<<elements;
217     file.close();
218   }
```

4.1.1.8 static void common::writeVectorToFile8 (uint8_t *& *vec,* int *length,* std::string *filename*) [inline], [static]

Writes an 8-bit unsigned integer vector to a .txt file.

Takes a vector of length 'length' and file path as arguments

Definition at line 223 of file commonheader.hpp.

```
224   {
225     std::ofstream file(filename);
226     if(!file)
227     {
228       cout<<"\nCould not create "<<filename<<"\nExiting...";
229       exit(0);
230     }
231
232     std::string elements = std::string("");
233     for(int i = 0; i < length; ++i)
234     {
235       elements.append(std::to_string(vec[i]));
236       elements.append("\n");
237     }
238
239     file<<elements;
240     file.close();
241   }
```

4.1.1.9 static void common::printArray8 (uint8_t *& *arr*, int *length*) [inline], [static]

Prints an 8-bit unsigned integer array of length 'length'.

Takes the array and its length as arguments

Definition at line 246 of file commonheader.hpp.

Referenced by main().

```
247   {
248     for(int i = 0; i < length; ++i)
249     {
250       printf(" %d",arr[i]);
251     }
252   }
```

Here is the caller graph for this function:

4.1.1.10 static void common::printArray16 (uint16_t *& *arr*, int *length*) [inline], [static]

Prints a 16-bit unsigned integer array of length 'length'.

Takes the array and its length as arguments

Definition at line 257 of file commonheader.hpp.

```
258   {
259     for(int i = 0; i < length; ++i)
260     {
261       printf(" %d",arr[i]);
262     }
263   }
```

4.1.1.11 static void common::printArray32 (uint32_t *& *arr,* int *length*) [inline], [static]

Prints a 32-bit unsigned integer array of length 'length'.

Takes the array and its length as arguments

Definition at line 268 of file commonheader.hpp.

Referenced by main().

```
269   {
270     for(int i = 0; i < length; ++i)
271     {
272       printf(" %d",arr[i]);
273     }
274   }
```

Here is the caller graph for this function:

4.1.1.12 static void common::printArrayDouble (double *& *arr,* int *length*) [inline], [static]

Prints a double array of length 'length'.

Takes the array and its length as arguments

Definition at line 279 of file commonheader.hpp.

```
280   {
281     for(int i = 0; i < length; ++i)
282     {
283       printf(" %f",arr[i]);
284     }
285   }
```

4.1.1.13 static uint32_t common::getRandomUnsignedInteger32 (uint32_t *lower_bound,* uint32_t *upper_bound*) [inline], [static]

Returns a random 32-bit unsigned integer within a range of (lower_bound,upper_bound).

Takes the lower_bound and upper_bound as arguments

Definition at line 305 of file commonheader.hpp.

```
306   {
307       //cout<<"\nIn getSeed";
308       std::random_device r;
309       std::seed_seq seed{r(), r(), r(), r(), r(), r(), r(), r()};
310       mt19937 seeder(seed);
311       uniform_int_distribution<uint32_t> intGen(lower_bound, upper_bound);
312       uint32_t alpha=intGen(seeder);
313       return alpha;
314   }
```

4.1.1.14 static int common::getRandomInteger (int *lower_bound,* int *upper_bound*) [inline],[static]

Returns a random 32-bit signed integer within a range of (lower_bound,upper_bound).

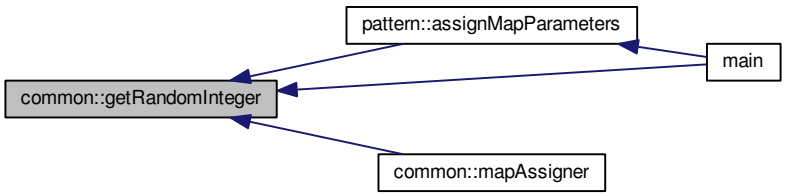
Takes the lower_bound and upper_bound as arguments

Definition at line 320 of file commonheader.hpp.

Referenced by pattern::assignMapParameters(), main(), and mapAssigner().

```
321   {
322       //cout<<"\nIn getSeed";
323       std::random_device r;
324       std::seed_seq seed{r(), r(), r(), r(), r(), r(), r(), r()};
325       mt19937 seeder(seed);
326       uniform_int_distribution<int> intGen(lower_bound, upper_bound);
327       uint32_t alpha=intGen(seeder);
328       return alpha;
329   }
```

Here is the caller graph for this function:



4.1.1.15 static uint8_t common::getRandomUnsignedInteger8 (uint8_t *lower_bound,* uint8_t *upper_bound*) [inline], [static]

Returns a random 8-bit unsigned integer within a range of (lower_bound,upper_bound).

Takes the lower_bound and upper_bound as arguments

Definition at line 291 of file commonheader.hpp.

```
292   {
293       std::random_device r;
294       std::seed_seq seed{r(), r(), r(), r(), r(), r(), r(), r()};
295       mt19937 seeder(seed);
296       uniform_int_distribution<uint8_t> intGen(lower_bound, upper_bound);
297       uint8_t alpha=intGen(seeder);
298       return alpha;
299   }
```

4.1.1.16 static double common::getRandomDouble (double *lower_limit,* double *upper_limit*) [inline],[static]

Returns a random double within a range of (lower_bound,upper_bound).

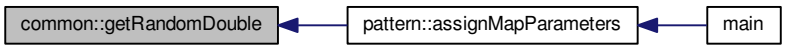
Takes the lower_bound and upper_bound as arguments

Definition at line 335 of file commonheader.hpp.

Referenced by pattern::assignMapParameters().

```
336   {
337      std::random_device r;
338      std::seed_seq seed{r(), r(), r(), r(), r(), r(), r(), r()};
339      mt19937 seeder(seed);
340      uniform_real_distribution<double> realGen(lower_limit,
      upper_limit);
341      auto randnum=realGen(seeder);
342      return randnum;
343   }
```

Here is the caller graph for this function:



4.1.1.17 static config::ChaoticMap common::mapAssigner (int *lower_limit,* int *upper_limit*) [inline],[static]

Returns a value of type ChaoticMap within a range of (lower_limit,upper_limit).

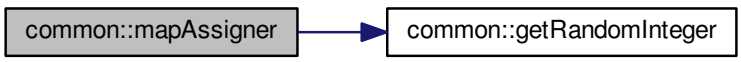
Takes the lower_limit and upper_limit as arguments

Definition at line 349 of file commonheader.hpp.

References getRandomInteger().

```
350   {
351     config::ChaoticMap chaotic_map;
352     chaotic_map = (config::ChaoticMap)getRandomInteger(
      lower_limit,upper_limit);
353     return chaotic_map;
354   }
```

Here is the call graph for this function:



4.1.1.18 static void common::rowColLUTGen (uint32_t *& *rowSwapLUT,* uint32_t *& *rowRandVec,* uint32_t *& *colSwapLUT,* uint32_t *& *colRandVec,* uint32_t *m,* uint32_t *n*) [inline], [static]

Generates shuffled row and column Lookup Tables using Fisher - Yates Shuffle for row and column rotation or swapping.

Takes two 1D vectors of length N X M and two vectors of length M-1 and N-1

Definition at line 359 of file commonheader.hpp.

```
360   {
361
362     int jCol=0,jRow=0;
363     for(int i = m - 1; i > 0; i--)
364     {
365       jRow = rowRandVec[i] % i;
366       std::swap(rowSwapLUT[i],rowSwapLUT[jRow]);
367     }
368
369     for(int i = n - 1; i > 0; i--)
370     {
371       jCol = colRandVec[i] % i;
372       std::swap(colSwapLUT[i],colSwapLUT[jCol]);
373     }
374   }
```

4.1.1.19 static void common::swapLUT (uint32_t *& *swapLUT,* uint32_t * *randVec,* uint32_t *m*) [inline], [static]

Shuffles the Lookup Table used to shuffle chaotic map choices array.

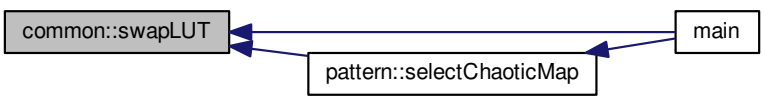
Takes a 1D vector of length M-1 and a 1D vector of length N X M and M as arguments

Definition at line 379 of file commonheader.hpp.

Referenced by main(), and pattern::selectChaoticMap().

```
380   {
381
382     int jLUT=0;
383     for(int i = m - 1; i > 0; i--)
384     {
385       jLUT = randVec[i] % i;
386       std::swap(swapLUT[i],swapLUT[jLUT]);
387     }
388
389   }
```

Here is the caller graph for this function:



4.1.1.20 static void common::genLUTVec (uint32_t *& *lut_vec,* uint32_t *n*) [inline],[static]

Generates a Lookup Table with values from 0 to n - 1 in ascending order for row and column swapping or rotating.

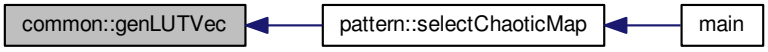
Takes a vector of length N and its length as arguments

Definition at line 394 of file commonheader.hpp.

Referenced by pattern::selectChaoticMap().

```
395   {
396     for(int i = 0; i < n; ++i)
397     {
398       lut_vec[i] = i;
399     }
400   }
```

Here is the caller graph for this function:



4.1.1.21 static void common::genMapLUTVec (uint32_t *& *lut_vec,* uint32_t *n*) [inline],[static]

Generates a Lookup Table with values from 1 to n in ascending order for shuffling the chaotic map choices array.

Takes a vector of length N and its length as arguments

Definition at line 405 of file commonheader.hpp.

Referenced by main().

```
406   {
407     int i = 0;
408     for(i = 0; i < n; ++i)
409     {
410       lut_vec[i] = i + 1;
411     }
412   }
```

Here is the caller graph for this function:

4.1.1.22 static std::string common::getFileNameFromPath (std::string *filename*) [inline],[static]

Gets the file name from the given file path.

Takes the file path as an argument

Definition at line 418 of file commonheader.hpp.

Referenced by main().

```
419   {
420     const size_t last_slash_idx = filename.find_last_of("\\/");
421     if (std::string::npos != last_slash_idx)
422     {
423       filename.erase(0, last_slash_idx + 1);
424     }
425
426     // Remove extension if present.
427     const size_t period_idx = filename.rfind('.');
428     if (std::string::npos != period_idx)
429     {
430       filename.erase(period_idx);
431     }
432
433     return filename;
434   }
```

Here is the caller graph for this function:

4.1.1.23 static std::string common::sha256_hash_string (unsigned char *hash[SHA256_DIGEST_LENGTH]*) [inline], [static]

Converts an 8-bit unsigned char SHA256 array into a SHA256 std::string.

Takes the 8-bit unsigned char hash array of length 64 as an argument

Definition at line 440 of file commonheader.hpp.

Referenced by calc_sha256().

```
441   {
442
443
444     stringstream ss;
445     for(int i = 0; i < SHA256_DIGEST_LENGTH; i++)
446     {
447         ss << hex << setw(2) << setfill('0') << (int)hash[i];
448     }
449
450     return ss.str();
451
452
453   }
```

Here is the caller graph for this function:

4.1.1.24 static std::string common::calc_sha256 (const char * *path*) [inline], [static]

Calculates SHA256 Hash of a given file.

Takes the file path as an argument

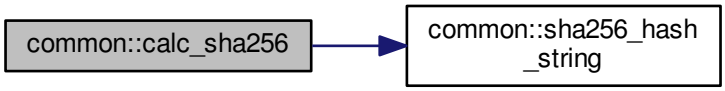
Definition at line 458 of file commonheader.hpp.

References sha256_hash_string().

Referenced by main().

```
459   {
460     FILE* file = fopen(path,"rb");
461
462     if(file==NULL)
463     {
464         printf("\n File Not found.\n Exiting...");
465         exit(0);
466     }
467
468     unsigned char hash[SHA256_DIGEST_LENGTH];
469     SHA256_CTX sha256;
470     SHA256_Init(&sha256);
471     const int bufSize = 32768;
472     char* buffer = (char*)malloc(bufSize);
473     int bytesRead = 0;
474
475     if(buffer==NULL)
476     {
477         printf("\n File Not found.\n Exiting...");
478         exit(0);
479     }
480
481     while((bytesRead = fread(buffer, 1, bufSize, file)))
482     {
483         SHA256_Update(&sha256, buffer, bytesRead);
484     }
485
486     SHA256_Final(hash, &sha256);
487
488     std::string hash_final = sha256_hash_string(hash);
489     cout<<"\nSHA256 hash of "<<path<<" is "<<hash_final;
490
491     fclose(file);
492     return hash_final;
493   }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.1.25 static void common::checkImageVectors (uint8_t * *plain_img_vec,* uint8_t * *decrypted_img_vec,* uint32_t *total*) [inline], [static]

Finds differences between two image vectors.

Takes two image vectors of length total and the length total as arguments

Definition at line 498 of file commonheader.hpp.

```
499   {
500     int cnt=0;
501     for(int i=0; i < total; ++i)
502     {
503       if(decrypted_img_vec[i]-plain_img_vec[i]!=0)
504       {
505         ++cnt;
506       }
507
508     }
509     printf("\nNumber of vector differences= %d",cnt);
510
511   }
```

4.1.1.26 static bool common::checkImages (cv::Mat *image_1,* cv::Mat *image_2*) [inline], [static]

Finds differences between two 2D N x M images.

Takes two 2D N X M images as arguments

Definition at line 516 of file commonheader.hpp.

Referenced by main().

```
517   {
518     if(image_1.rows != image_2.rows or image_1.cols != image_2.cols)
519     {
520       cout<<"\nCould not comapare images\nExiting...";
521       exit(0);
522     }
523
524     uint8_t difference = 0;
525     uint32_t count_differences = 0;
526     for(int i = 0; i < image_1.rows; ++i)
527     {
528       for(int j = 0; j < image_1.cols; ++j)
529       {
530         for(int k = 0; k < image_1.channels(); ++k)
531         {
532           difference = image_1.at<Vec3b>(i,j)[k] - image_2.at<Vec3b>(i,j)[k];
```

```
533             if(difference != 0)
534             {
535               ++count_differences;
536             }
537           }
538         }
539       }
540
541       if(count_differences != 0)
542       {
543         cout<<"\nDifferences between decrypted image and plain image = "<<count_differences;
544         return 0;
545       }
546
547       else
548       {
549         cout<<"\nDifferences between decrypted image and plain image = "<<count_differences;
550         return 1;
551       }
552     }
```

Here is the caller graph for this function:

4.2 config Namespace Reference

Classes

- struct lalm

  Structure to store Logistic Adjusted Logistic Map parameters.
- struct lasm

  Structure to store Logistic Adjusted Sine Map parameters.
- struct lm

  Structure to store Logistic Map parameters.
- struct lma

  Structure to store Advanced Logistic Map parameters.
- struct mt

  Structure to store Mersenne Twister parameters.
- struct slmm

  Structure to store Sine Logistic Modulation Map parameters.

Enumerations

- enum ChaoticMap {
  ChaoticMap::TwoDLogisticMap = 1, ChaoticMap::TwoDLogisticMapAdvanced, ChaoticMap::TwoDLogistic↩
  AdjustedSineMap, ChaoticMap::TwoDSineLogisticModulationMap,
  ChaoticMap::TwoDLogisticAdjustedLogisticMap }

  Enumeration class used to denote Chaotic Map choices.

Variables

- int lower_limit = 1

  Range of pseudorandom values to be produced by Mersenne Twister.
- int upper_limit = (rows ∗ cols ∗ 3) + 1
- std::string parameters_file = "parameters"

  The file path of the parameter file where all parameters are stored.
- std::string binary_extension = ".bin"
- std::string parameters_file_path = parameters_file + binary_extension
- char ∗ constant_parameters_file_path = const_cast<char∗>(parameters_file_path.c_str())

  char∗ constant strings for use with standard C file handling functions

4.2.1 Enumeration Type Documentation

4.2.1.1 enum config::ChaoticMap [strong]

Enumeration class used to denote Chaotic Map choices.

Enumerator

TwoDLogisticMap

TwoDLogisticMapAdvanced

TwoDLogisticAdjustedSineMap

TwoDSineLogisticModulationMap

TwoDLogisticAdjustedLogisticMap

Definition at line 105 of file config.hpp.

```
106   {
107     TwoDLogisticMap = 1,
108     TwoDLogisticMapAdvanced,
109     TwoDLogisticAdjustedSineMap,
110     TwoDSineLogisticModulationMap,
111     TwoDLogisticAdjustedLogisticMap
112   };
```

4.2.2 Variable Documentation

4.2.2.1 int config::lower_limit = 1

Range of pseudorandom values to be produced by Mersenne Twister.

Definition at line 99 of file config.hpp.

Referenced by pattern::selectChaoticMap().

4.2.2.2 int config::upper_limit = (rows ∗ cols ∗ 3) + 1

Definition at line 100 of file config.hpp.

Referenced by pattern::selectChaoticMap().

4.2.2.3 std::string config::parameters_file = "parameters"

The file path of the parameter file where all parameters are stored.

Definition at line 184 of file config.hpp.

4.2.2.4 std::string config::binary_extension = ".bin"

Definition at line 185 of file config.hpp.

4.2.2.5 std::string config::parameters_file_path = parameters_file + binary_extension

Definition at line 186 of file config.hpp.

4.2.2.6 char* config::constant_parameters_file_path = const_cast<char*>(parameters_file_path.c_str())

char* constant strings for use with standard C file handling functions

Definition at line 191 of file config.hpp.

Referenced by main(), and pattern::rwMapParameters().

4.3 pattern Namespace Reference

This header file contains chaotic map functions to generate pseudorandom sequences and those functions associated to them.

Functions

- static void twodLogisticMapAdvanced (double *&x, double *&y, uint32_t *&random_array, double myu1, double myu2, double lambda1, double lambda2, uint32_t number)

  Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLMA.
- static void twodLogisticAdjustedSineMap (double *&x, double *&y, uint32_t *&random_array, double myu, uint32_t total)

  Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLASM.
- static void MTSequence (uint32_t *&random_array, uint32_t total, int lower_limit, int upper_limit, int seed)

  Produces a pseudorandom sequence of 32-bit unsigned integers using Mersenne Twister.
- static void twodSineLogisticModulationMap (double *&x, double *&y, uint32_t *&random_array, double alpha, double beta, uint32_t total)

  Produces a pseudorandom sequence of 32-bit unsigned integers using 2DSLMM.
- static void twodLogisticAdjustedLogisticMap (double *&x, double *&y, double *&x_bar, double *&y_bar, uint32_t *&random_array, double myu, uint32_t total)

  Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLALM.
- static void twodLogisticMap (double *&x, double *&y, uint32_t *&random_array, double r, uint32_t total)

  Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLALM.

- static void initializeMapParameters (config::lm lm_parameters[ ], config::lma lma_parameters[ ], config::slmm slmm_parameters[ ], config::lasm lasm_parameters[ ], config::lalm lalm_parameters[ ], config::mt mt_parameters[ ], config::ChaoticMap map, int number_of_rounds)

  Initializes all chaotic map parameters to zero.
- static void assignMapParameters (config::lm lm_parameters[ ], config::lma lma_parameters[ ], config::slmm slmm_parameters[ ], config::lasm lasm_parameters[ ], config::lalm lalm_parameters[ ], config::mt mt_parameters[ ], config::ChaoticMap map, int number_of_rounds)

  Initializes all chaotic map parameters to random values.
- static void displayMapParameters (config::lm lm_parameters[ ], config::lma lma_parameters[ ], config::slmm slmm_parameters[ ], config::lasm lasm_parameters[ ], config::lalm lalm_parameters[ ], config::mt mt_parameters[ ], config::ChaoticMap map, int number_of_rounds)

  Displays all chaotic map parameters.
- static long rwMapParameters (config::lm lm_parameters[ ], config::lma lma_parameters[ ], config::slmm slmm_parameters[ ], config::lasm lasm_parameters[ ], config::lalm lalm_parameters[ ], config::mt mt_parameters[ ], config::ChaoticMap map, FILE *outfile, const char *mode, long ptr_position, int number_of_rounds)

  Reads or writes parameters of chosen chaotic map.
- static void selectChaoticMap (config::lm lm_parameters[ ], config::lma lma_parameters[ ], config::slmm slmm_parameters[ ], config::lasm lasm_parameters[ ], config::lalm lalm_parameters[ ], config::mt mt_parameters[ ], double *x, double *y, double *x_bar, double *y_bar, uint32_t *&random_array, uint32_t *&lut_vec, config::ChaoticMap map, int iteration, uint32_t m, uint32_t random_array_length)

  Selects the chaotic map according to given chaotic map choice.

4.3.1 Detailed Description

This header file contains chaotic map functions to generate pseudorandom sequences and those functions associated to them.

4.3.2 Function Documentation

4.3.2.1 static void pattern::twodLogisticMapAdvanced (double *& *x*, double *& *y*, uint32_t *& *random_array*, double *myu1*, double *myu2*, double *lambda1*, double *lambda2*, uint32_t *number*) [inline], [static]

Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLMA.

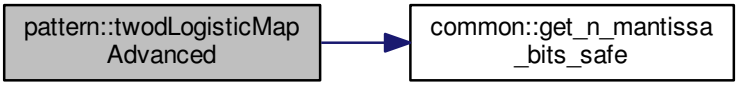
Definition at line 31 of file pattern.hpp.

References common::get_n_mantissa_bits_safe(), and NUMBER_OF_BITS.

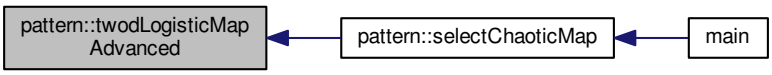
Referenced by selectChaoticMap().

```
32   {
33
34     int i = 0;
35     for(i = 0; i < number - 1; ++i)
36     {
37
38       x[i + 1] = x[i] * myu1 * (1 - x[i]) + lambda1 * (y[i] * y[i]);
39       y[i + 1] = y[i] * myu2 * (1 - y[i]) + lambda2 * ((x[i] * x[i]) + x[i] * y[i]);
40     }
41
42     for(int i = 0; i < number; ++i)
43     {
44       random_array[i] = common::get_n_mantissa_bits_safe(x[i],
     NUMBER_OF_BITS);
45     }
46   }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.2.2 static void pattern::twodLogisticAdjustedSineMap (double *& *x*, double *& *y*, uint32_t *& *random_array*, double *myu*, uint32_t *total*) [inline], [static]

Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLASM.

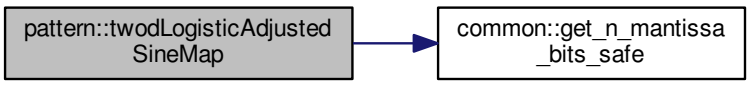
Definition at line 51 of file pattern.hpp.

References common::get_n_mantissa_bits_safe(), and NUMBER_OF_BITS.

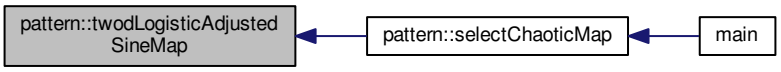
Referenced by selectChaoticMap().

```
52   {
53
54     int i=0;
55
56     for(i = 0; i < (total) - 1; ++i)
57     {
58       x[i + 1] = sin(M_PI * myu * (y[i] + 3) * x[i] * (1 - x[i]));
59       y[i + 1] = sin(M_PI * myu * (x[i + 1] + 3) * y[i] * (1 - y[i]));
60     }
61
62     for(int i = 0; i < total; ++i)
63     {
64       random_array[i] = common::get_n_mantissa_bits_safe(x[i],
     NUMBER_OF_BITS);
65     }
66   }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.2.3 static void pattern::MTSequence (uint32_t *& *random_array,* uint32_t *total,* int *lower_limit,* int *upper_limit,* int *seed*) [inline], [static]

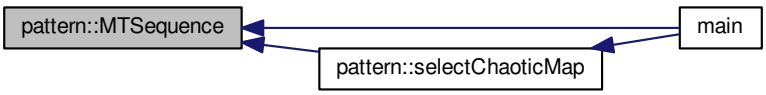
Produces a pseudorandom sequence of 32-bit unsigned integers using Mersenne Twister.

Definition at line 72 of file pattern.hpp.

Referenced by main(), and selectChaoticMap().

```
73   {
74     std::mt19937 seeder(seed);
75
76     std::uniform_int_distribution<int> intGen(lower_limit,upper_limit);
77
78     for (size_t i = 0; i < total; ++i)
79     {
80         auto random_number = intGen(seeder);
81         random_array[i]=(uint32_t)random_number;
82     }
83   }
```

Here is the caller graph for this function:



4.3.2.4 static void pattern::twodSineLogisticModulationMap (double *& *x,* double *& *y,* uint32_t *& *random_array,* double *alpha,* double *beta,* uint32_t *total*) [inline], [static]

Produces a pseudorandom sequence of 32-bit unsigned integers using 2DSLMM.

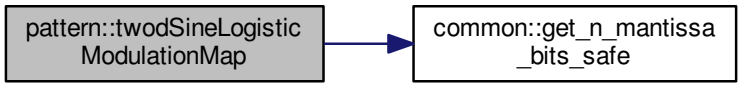
Definition at line 89 of file pattern.hpp.

References common::get_n_mantissa_bits_safe(), and NUMBER_OF_BITS.

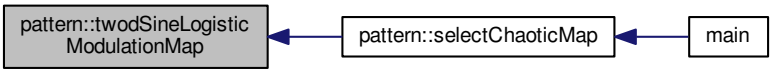
Referenced by selectChaoticMap().

```
90   {
91
92     for(int i = 0; i < (total) - 1; ++i)
93     {
94       x[i + 1] = alpha * (sin(M_PI * y[i]) + beta) * x[i] * (1 - x[i]);
95       y[i + 1] = alpha * (sin(M_PI * x[i + 1]) + beta) * y[i] * (1 - y[i]);
96     }
97
98     for(int i = 0; i < total; ++i)
99     {
100       random_array[i] = common::get_n_mantissa_bits_safe(x[i],
      NUMBER_OF_BITS);
101     }
102   }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.2.5 static void pattern::twodLogisticAdjustedLogisticMap (double *& *x*, double *& *y*, double *& *x_bar*, double *& *y_bar*, uint32_t *& *random_array*, double *myu*, uint32_t *total*) [inline], [static]

Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLALM.

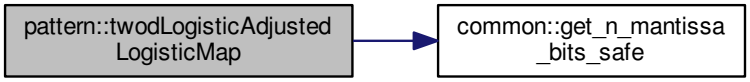
Definition at line 108 of file pattern.hpp.

References common::get_n_mantissa_bits_safe(), and NUMBER_OF_BITS.

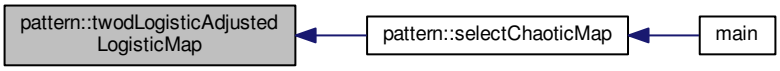
Referenced by selectChaoticMap().

```
109   {
110
111     for(uint32_t i = 0; i < total - 1; ++i)
112     {
113       x_bar[i + 1] = myu * (y[i] * 3) * x[i] * (1 - x[i]);
114       x[i + 1] = 4 * x_bar[i + 1] * (1 - x_bar[i + 1]);
115       y_bar[i + 1] = myu * (x[i + 1] + 3) * y[i] * (1 - y[i]);
116       y[i + 1] = 4 * y_bar[i + 1] * (1 - y_bar[i + 1]);
117     }
118
119     for(int i = 0; i < total; ++i)
120     {
121       random_array[i] = common::get_n_mantissa_bits_safe(x[i],
      NUMBER_OF_BITS);
122     }
123   }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.2.6 static void pattern::twodLogisticMap (double *& *x*, double *& *y*, uint32_t *& *random_array*, double *r*, uint32_t *total*) [inline], [static]

Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLALM.

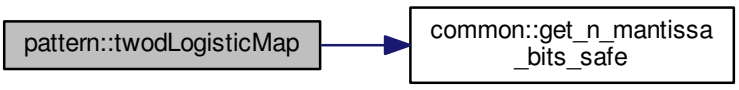
Definition at line 128 of file pattern.hpp.

References common::get_n_mantissa_bits_safe(), and NUMBER_OF_BITS.

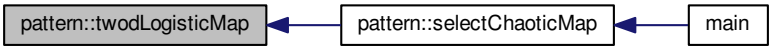
Referenced by selectChaoticMap().

```
129   {
130
131     for(uint32_t i = 0; i < (total) - 1; ++i)
132     {
133
134
135       x[i + 1] = r * ((3 * y[i]) + 1) * x[i] * (1 - x[i]);
136       y[i + 1] = r * ((3 * x[i + 1]) + 1) * y[i] * (1 - y[i]);
137     }
138
139     for(int i = 0; i < total; ++i)
140     {
141       random_array[i] = common::get_n_mantissa_bits_safe(x[i],
      NUMBER_OF_BITS);
142     }
143   }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.2.7 static void pattern::initializeMapParameters (config::lm *lm_parameters[ ]*, config::lma *lma_parameters[ ]*, config::slmm *slmm_parameters[ ]*, config::lasm *lasm_parameters[ ]*, config::lalm *lalm_parameters[ ]*, config::mt *mt_parameters[ ]*, config::ChaoticMap *map*, int *number_of_rounds*) [inline], [static]

Initializes all chaotic map parameters to zero.

Initializing all parameters to zero

Initializing all parameters to zero

Definition at line 148 of file pattern.hpp.

References config::slmm::alpha, config::slmm::beta, config::lma::lambda1, config::lma::lambda2, config::lalm::myu, config::lasm::myu, config::lma::myu1, config::lma::myu2, config::lm::r, config::mt::seed_1, config::lalm::x_init, config::lasm::x_init, config::slmm::x_init, config::lma::x_init, config::lm::x_init, config::lalm::y_init, config::lasm::y_init, config::slmm::y_init, config::lma::y_init, and config::lm::y_init.

Referenced by main().

```
149   {
150
154     for(int i = 0; i < number_of_rounds; ++i)
155     {
156       if(int(map) == 1)
157       {
158
159         lm_parameters[i].x_init = 0.00;
160         lm_parameters[i].y_init = 0.00;
161         lm_parameters[i].r = 0.00;
162       }
163
164       if(int(map) == 2)
165       {
166
167         lma_parameters[i].x_init = 0.00;
168         lma_parameters[i].y_init = 0.00;
169         lma_parameters[i].myu1 = 0.00;
170         lma_parameters[i].myu2 = 0.00;
171         lma_parameters[i].lambda1 = 0.00;
172         lma_parameters[i].lambda2 = 0.00;
173       }
174
175       if(int(map) == 3)
176       {
177
178         slmm_parameters[i].x_init = 0.00;
179         slmm_parameters[i].y_init = 0.00;
180         slmm_parameters[i].alpha = 0.00;
181         slmm_parameters[i].beta = 0.00;
182       }
183
184       if(int(map) == 4)
185       {
186
187         lasm_parameters[i].x_init = 0.00;
188         lasm_parameters[i].y_init = 0.00;
189         lasm_parameters[i].myu = 0.00;
```

```
190       }
191
192       if(int(map) == 5)
193       {
194
195         lalm_parameters[i].x_init = 0.00;
196         lalm_parameters[i].y_init = 0.00;
197         lalm_parameters[i].myu = 0.00;
198       }
199
200       if(int(map) == 6)
201       {
202
203         mt_parameters[i].seed_1 = 0;;
204       }
205     }
206   }
```

Here is the caller graph for this function:

4.3.2.8 static void pattern::assignMapParameters (config::lm *lm_parameters[ ],* config::lma *lma_parameters[ ],* config::slmm *slmm_parameters[ ],* config::lasm *lasm_parameters[ ],* config::lalm *lalm_parameters[ ],* config::mt *mt_parameters[ ],* config::ChaoticMap *map,* int *number_of_rounds*) [inline], [static]

Initializes all chaotic map parameters to random values.

Definition at line 211 of file pattern.hpp.

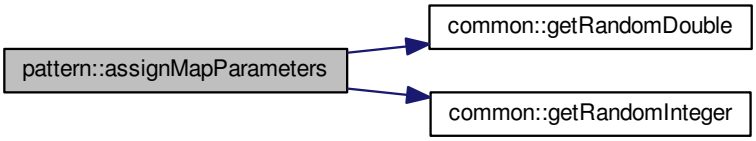
References config::slmm::alpha, ALPHA_LOWER_LIMIT, ALPHA_UPPER_LIMIT, config::slmm::beta, BETA_L↩
OWER_LIMIT, BETA_UPPER_LIMIT, common::getRandomDouble(), common::getRandomInteger(), config::lma↩
::lambda1, LAMBDA1_LOWER_LIMIT, LAMBDA1_UPPER_LIMIT, config::lma::lambda2, LAMBDA2_LOWER_L↩
IMIT, LAMBDA2_UPPER_LIMIT, config::lalm::myu, config::lasm::myu, config::lma::myu1, MYU1_LOWER_LIM↩
IT, MYU1_UPPER_LIMIT, config::lma::myu2, MYU2_LOWER_LIMIT, MYU2_UPPER_LIMIT, MYU_LOWER_LI↩
MIT, MYU_UPPER_LIMIT, config::lm::r, R_LOWER_LIMIT, R_UPPER_LIMIT, config::mt::seed_1, SEED_LOW↩
ER_LIMIT, SEED_UPPER_LIMIT, config::lalm::x_init, config::lasm::x_init, config::slmm::x_init, config::lma::x_init, config::lm::x_init, X_LOWER_LIMIT, X_UPPER_LIMIT, config::lalm::y_init, config::lasm::y_init, config::slmm::y_↩
init, config::lma::y_init, config::lm::y_init, Y_LOWER_LIMIT, and Y_UPPER_LIMIT.

Referenced by main().

```
212   {
213
214     for(int i = 0; i < number_of_rounds; ++i)
215     {
216       if(int(map) == 1)
217       {
218
219         lm_parameters[i].x_init = common::getRandomDouble(
      X_LOWER_LIMIT,X_UPPER_LIMIT);
220         lm_parameters[i].y_init = common::getRandomDouble(
      Y_LOWER_LIMIT,Y_UPPER_LIMIT);
221         lm_parameters[i].r = common::getRandomDouble(
      R_LOWER_LIMIT,R_UPPER_LIMIT);
222       }
```

```
223
224       if(int(map) == 2)
225       {
226
227         lma_parameters[i].x_init = common::getRandomDouble(
      X_LOWER_LIMIT,X_UPPER_LIMIT);
228         lma_parameters[i].y_init = common::getRandomDouble(
      Y_LOWER_LIMIT,Y_UPPER_LIMIT);
229         lma_parameters[i].myu1 =  common::getRandomDouble(
      MYU1_LOWER_LIMIT,MYU1_UPPER_LIMIT);
230         lma_parameters[i].myu2 = common::getRandomDouble(
      MYU2_LOWER_LIMIT,MYU2_UPPER_LIMIT);
231         lma_parameters[i].lambda1 = common::getRandomDouble(
      LAMBDA1_LOWER_LIMIT,LAMBDA1_UPPER_LIMIT);
232         lma_parameters[i].lambda2 = common::getRandomDouble(
      LAMBDA2_LOWER_LIMIT,LAMBDA2_UPPER_LIMIT);
233       }
234
235       if(int(map) == 3)
236       {
237
238         slmm_parameters[i].x_init = common::getRandomDouble(
      X_LOWER_LIMIT,X_UPPER_LIMIT);
239         slmm_parameters[i].y_init = common::getRandomDouble(
      Y_LOWER_LIMIT,Y_UPPER_LIMIT);
240         slmm_parameters[i].alpha = common::getRandomDouble(
      ALPHA_LOWER_LIMIT,ALPHA_UPPER_LIMIT);
241         slmm_parameters[i].beta = common::getRandomDouble(
      BETA_LOWER_LIMIT,BETA_UPPER_LIMIT);
242       }
243
244       if(int(map) == 4)
245       {
246
247         lasm_parameters[i].x_init = common::getRandomDouble(
      X_LOWER_LIMIT,X_UPPER_LIMIT);
248         lasm_parameters[i].y_init = common::getRandomDouble(
      Y_LOWER_LIMIT,Y_UPPER_LIMIT);
249         lasm_parameters[i].myu = common::getRandomDouble(
      MYU_LOWER_LIMIT,MYU_UPPER_LIMIT);
250       }
251
252       if(int(map) == 5)
253       {
254
255         lalm_parameters[i].x_init = common::getRandomDouble(
      X_LOWER_LIMIT,X_UPPER_LIMIT);
256         lalm_parameters[i].y_init = common::getRandomDouble(
      Y_LOWER_LIMIT,Y_UPPER_LIMIT);
257         lalm_parameters[i].myu = common::getRandomDouble(
      MYU_LOWER_LIMIT,MYU_UPPER_LIMIT);
258       }
259
260       if(int(map) == 6)
261       {
262
263         mt_parameters[i].seed_1 = common::getRandomInteger(
      SEED_LOWER_LIMIT,SEED_UPPER_LIMIT);
264       }
265     }
266   }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.3.2.9 static void pattern::displayMapParameters (config::lm *lm_parameters[ ]*, config::lma *lma_parameters[ ]*, config::slmm *slmm_parameters[ ]*, config::lasm *lasm_parameters[ ]*, config::lalm *lalm_parameters[ ]*, config::mt *mt_parameters[ ]*, config::ChaoticMap *map*, int *number_of_rounds*) [inline], [static]

Displays all chaotic map parameters.

Definition at line 272 of file pattern.hpp.

Referenced by main().

```
273   {
274
275     for(int i = 0; i < number_of_rounds; ++i)
276     {
277       if(int(map) == 1)
278       {
279         cout<<"\n\nROUND "<<i;
280         printf("\nlm_parameters.x_init = %f",lm_parameters[i].x_init);
281         printf("\nlm_parameters.y_init = %f",lm_parameters[i].y_init);
282         printf("\nlm_parameters.r = %f",lm_parameters[i].r);
283       }
284
285       if(int(map) == 2)
286       {
287         cout<<"\n\nROUND "<<i;
288         printf("\nlma_parameters.x_init = %f",lma_parameters[i].x_init);
289         printf("\nlma_parameters.y_init = %f",lma_parameters[i].y_init);
290         printf("\nlma_parameters.myu1 = %f",lma_parameters[i].myu1);
291         printf("\nlma_parameters.myu2 = %f",lma_parameters[i].myu2);
292         printf("\nlma_parameters.lambda1 = %f",lma_parameters[i].lambda1);
293         printf("\nlma_parameters.lambda2 = %f",lma_parameters[i].lambda2);
294       }
295
296       if(int(map) == 3)
297       {
298         cout<<"\n\nROUND "<<i;
299         printf("\nslmm_parameters.x_init = %f",slmm_parameters[i].x_init);
300         printf("\nslmm_parameters.y_init = %f",slmm_parameters[i].y_init);
301         printf("\nslmm_parameters.alpha = %f",slmm_parameters[i].alpha);
302         printf("\nslmm_parameters.beta = %f",slmm_parameters[i].beta);
303       }
304
305       if(int(map) == 4)
306       {
307         cout<<"\n\nROUND "<<i;
308         printf("\nlasm parameters.x_init = %f",lasm_parameters[i].x_init);
309         printf("\nlasm parameters.y_init = %f",lasm_parameters[i].y_init);
310         printf("\nlasm parameters.myu = %f",lasm_parameters[i].myu);
311       }
312
313       if(int(map) == 5)
314       {
315         cout<<"\n\nROUND "<<i;
316         printf("\nlalm parameters.x_init = %f",lalm_parameters[i].x_init);
317         printf("\nlalm parameters.y_init = %f",lalm_parameters[i].y_init);
318         printf("\nlalm parameters.myu = %f",lalm_parameters[i].myu);
319       }
320
321       if(int(map) == 6)
```

```
322       {
323         cout<<"\n\nROUND "<<i;
324         printf("\nmt_parameters.seed_1 = %d",mt_parameters[i].seed_1);
325       }
326     }
327   }
```

Here is the caller graph for this function:

4.3.2.10 static long pattern::rwMapParameters (config::lm *lm_parameters[ ],* config::lma *lma_parameters[ ],* config::slmm *slmm_parameters[ ],* config::lasm *lasm_parameters[ ],* config::lalm *lalm_parameters[ ],* config::mt *mt_parameters[ ],* config::ChaoticMap *map,* FILE * *outfile,* const char * *mode,* long *ptr_position,* int *number_of_rounds*) [inline], [static]

Reads or writes parameters of chosen chaotic map.

Definition at line 332 of file pattern.hpp.

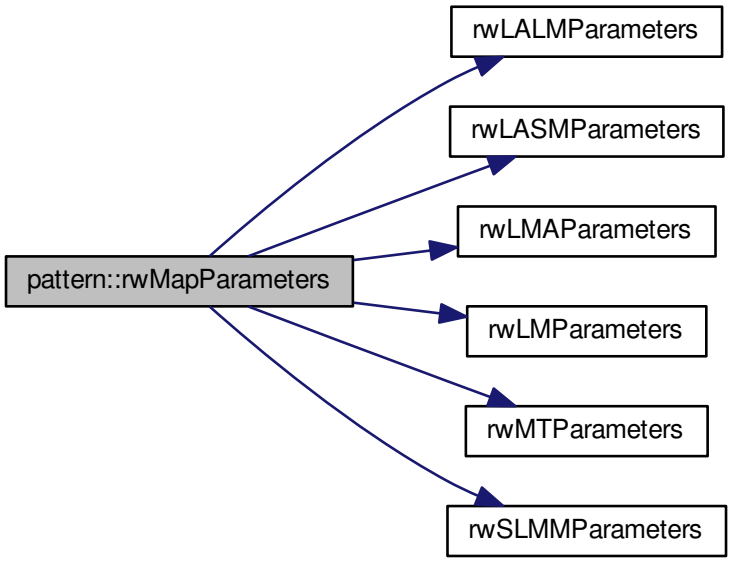
References config::constant_parameters_file_path, rwLALMParameters(), rwLASMParameters(), rwLMA↩Parameters(), rwLMParameters(), rwMTParameters(), and rwSLMMParameters().

Referenced by main().

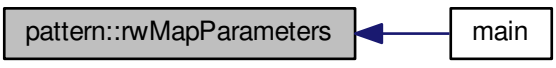
```
333   {
334     if(int(map) == 1)
335     {
336       ptr_position = rwLMParameters(outfile,
      config::constant_parameters_file_path,mode,lm_parameters,0,
      number_of_rounds,ptr_position);
337       return ptr_position;
338     }
339
340     if(int(map) == 2)
341     {
342       ptr_position = rwLMAParameters(outfile,
      config::constant_parameters_file_path,mode,lma_parameters,0,
      number_of_rounds,ptr_position);
343       return ptr_position;
344     }
345
346     if(int(map) == 3)
347     {
348       ptr_position = rwSLMMParameters(outfile,
      config::constant_parameters_file_path,mode,slmm_parameters,0,
      number_of_rounds,ptr_position);
349       return ptr_position;
350     }
351
352     if(int(map) == 4)
353     {
354       ptr_position = rwLASMParameters(outfile,
      config::constant_parameters_file_path,mode,lasm_parameters,0,
      number_of_rounds,ptr_position);
355       return ptr_position;
356     }
357
358     if(int(map) == 5)
```

```
359     {
360       ptr_position = rwLALMParameters(outfile,
      config::constant_parameters_file_path,mode,lalm_parameters,0,
      number_of_rounds,ptr_position);
361       return ptr_position;
362     }
363
364     if(int(map) == 6)
365     {
366       ptr_position = rwMTParameters(outfile,
      config::constant_parameters_file_path,mode,mt_parameters,0,
      number_of_rounds,ptr_position);
367       return ptr_position;
368     }
369   }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.2.11 static void pattern::selectChaoticMap (config::lm *lm_parameters[ ],* config::lma *lma_parameters[ ],* config::slmm *slmm_parameters[ ],* config::lasm *lasm_parameters[ ],* config::lalm *lalm_parameters[ ],* config::mt *mt_parameters[ ],* double * *x,* double * *y,* double * *x_bar,* double * *y_bar,* uint32_t *&* *random_array,* uint32_t *&* *lut_vec,* config::ChaoticMap *map,* int *iteration,* uint32_t *m,* uint32_t *random_array_length*) [inline], [static]

Selects the chaotic map according to given chaotic map choice.

Definition at line 374 of file pattern.hpp.

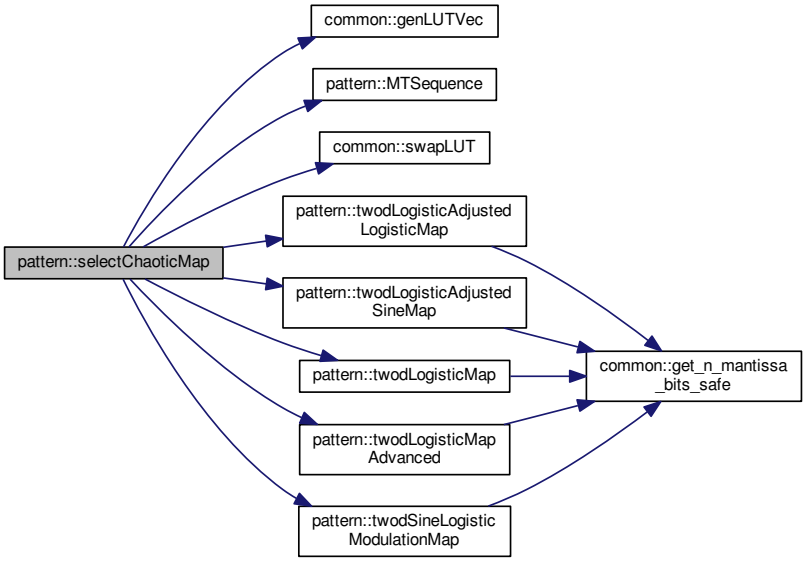
References common::genLUTVec(), config::lower_limit, MTSequence(), common::swapLUT(), twodLogisticAdjustedLogisticMap(), twodLogisticAdjustedSineMap(), twodLogisticMap(), twodLogisticMapAdvanced(), twodSineLogisticModulationMap(), config::upper_limit, config::lalm::x_init, config::lasm::x_init, config::slmm::x_init, config::lma::x_init, config::lm::x_init, config::lalm::y_init, config::lasm::y_init, config::slmm::y_init, config::lma::y_init, and config::lm::y_init.

Referenced by main().

```
375   {
376     if(int(map) == 1)
377     {
378       x[0] = lm_parameters[iteration].x_init;
379       y[0] = lm_parameters[iteration].y_init;
380       twodLogisticMap(x,y,random_array,lm_parameters[iteration].r,random_array_length);
381       common::genLUTVec(lut_vec,m);
382       common::swapLUT(lut_vec,random_array,m);
383     }
384
385     else if(int(map) == 2)
386     {
387       x[0] = lma_parameters[iteration].x_init;
388       y[0] = lma_parameters[iteration].y_init;
389       twodLogisticMapAdvanced(x,y,random_array,lma_parameters[iteration].myu1,
    lma_parameters[iteration].myu2,lma_parameters[iteration].lambda1,lma_parameters[iteration].lambda2,
    random_array_length);
390
391       common::genLUTVec(lut_vec,m);
392       common::swapLUT(lut_vec,random_array,m);
393     }
394
395     else if(int(map) == 3)
396     {
397
398       x[0] = slmm_parameters[iteration].x_init;
399       y[0] = slmm_parameters[iteration].y_init;
400       twodSineLogisticModulationMap(x,y,random_array,slmm_parameters[iteration
    ].alpha,slmm_parameters[iteration].beta,random_array_length);
401
402       common::genLUTVec(lut_vec,m);
403       common::swapLUT(lut_vec,random_array,m);
404
405     }
406
407     else if(int(map) == 4)
408     {
409       x[0] = lasm_parameters[iteration].x_init;
410       y[0] = lasm_parameters[iteration].y_init;
411       twodLogisticAdjustedSineMap(x,y,random_array,lasm_parameters[iteration].
    myu,random_array_length);
412
413       common::genLUTVec(lut_vec,m);
414       common::swapLUT(lut_vec,random_array,m);
415     }
416
417     else if(int(map) == 5)
418     {
419       x[0] = lalm_parameters[iteration].x_init;
420       y[0] = lalm_parameters[iteration].y_init;
421       twodLogisticAdjustedLogisticMap(x,y,x_bar,y_bar,random_array,
    lalm_parameters[iteration].myu,random_array_length);
422
423       common::genLUTVec(lut_vec,m);
424       common::swapLUT(lut_vec,random_array,m);
425     }
426
427     else if(int(map) == 6)
428     {
```

```
429         MTSequence(random_array,random_array_length,config::lower_limit,
    config::upper_limit,mt_parameters[iteration].seed_1);
430
431         common::genLUTVec(lut_vec,m);
432         common::swapLUT(lut_vec,random_array,m);
433
434       }
435
436     }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.4 serial Namespace Reference

This header file contains the Gray Level Transform function used for image diffusion.

Functions

- static void grayLevelTransform (uint8_t *&img_vec, uint32_t *random_array, uint32_t total)

  Diffuses image vector with pseudorandom sequence.

4.4.1 Detailed Description

This header file contains the Gray Level Transform function used for image diffusion.

4.4.2 Function Documentation

4.4.2.1 static void serial::grayLevelTransform (uint8_t *& *img_vec,* uint32_t * *random_array,* uint32_t *total*) [inline], [static]

Diffuses image vector with pseudorandom sequence.

Definition at line 15 of file serial.hpp.

Referenced by main().

```
16   {
17     int i = 0;
18     for(i = 0; i < total; ++i)
19     {
20       img_vec[i] = img_vec[i] ^ random_array[i];
21     }
22
23   }
```

Here is the caller graph for this function:

Chapter 5

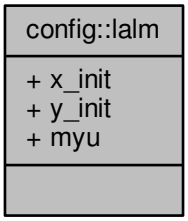
Class Documentation

5.1 config::lalm Struct Reference

Structure to store Logistic Adjusted Logistic Map parameters.

`#include <config.hpp>`

Collaboration diagram for config::lalm:



Public Attributes

- double x_init
- double y_init
- double myu

5.1.1 Detailed Description

Structure to store Logistic Adjusted Logistic Map parameters.

Definition at line 117 of file config.hpp.

5.1.2 Member Data Documentation

5.1.2.1 double config::lalm::x_init

Definition at line 119 of file config.hpp.

Referenced by pattern::assignMapParameters(), pattern::initializeMapParameters(), and pattern::selectChaotic↩ Map().

5.1.2.2 double config::lalm::y_init

Definition at line 120 of file config.hpp.

Referenced by pattern::assignMapParameters(), pattern::initializeMapParameters(), and pattern::selectChaotic↩ Map().

5.1.2.3 double config::lalm::myu

Definition at line 121 of file config.hpp.

Referenced by pattern::assignMapParameters(), and pattern::initializeMapParameters().

The documentation for this struct was generated from the following file:

- include/config.hpp

5.2 config::lasm Struct Reference

Structure to store Logistic Adjusted Sine Map parameters.

`#include <config.hpp>`

Collaboration diagram for config::lasm:

Public Attributes

- double x_init
- double y_init
- double myu

5.2.1 Detailed Description

Structure to store Logistic Adjusted Sine Map parameters.

Definition at line 128 of file config.hpp.

5.2.2 Member Data Documentation

5.2.2.1 double config::lasm::x_init

Definition at line 130 of file config.hpp.

Referenced by pattern::assignMapParameters(), pattern::initializeMapParameters(), and pattern::selectChaotic↩ Map().

5.2.2.2 double config::lasm::y_init

Definition at line 131 of file config.hpp.

Referenced by pattern::assignMapParameters(), pattern::initializeMapParameters(), and pattern::selectChaotic↩ Map().

5.2.2.3 double config::lasm::myu

Definition at line 132 of file config.hpp.

Referenced by pattern::assignMapParameters(), and pattern::initializeMapParameters().

The documentation for this struct was generated from the following file:

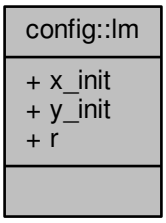
- include/config.hpp

5.3 config::lm Struct Reference

Structure to store Logistic Map parameters.

`#include <config.hpp>`

Collaboration diagram for config::lm:



Public Attributes

- double x_init
- double y_init
- double r

5.3.1 Detailed Description

Structure to store Logistic Map parameters.

Definition at line 165 of file config.hpp.

5.3.2 Member Data Documentation

5.3.2.1 double config::lm::x_init

Definition at line 167 of file config.hpp.

Referenced by pattern::assignMapParameters(), pattern::initializeMapParameters(), and pattern::selectChaotic↩ Map().

5.3.2.2 double config::lm::y_init

Definition at line 168 of file config.hpp.

Referenced by pattern::assignMapParameters(), pattern::initializeMapParameters(), and pattern::selectChaotic↩ Map().

5.3.2.3 double config::lm::r

Definition at line 169 of file config.hpp.

Referenced by pattern::assignMapParameters(), and pattern::initializeMapParameters().

The documentation for this struct was generated from the following file:

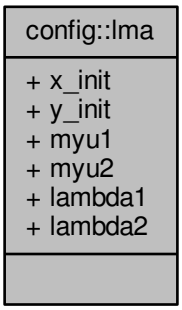
- include/config.hpp

5.4 config::lma Struct Reference

Structure to store Advanced Logistic Map parameters.

`#include <config.hpp>`

Collaboration diagram for config::lma:



Public Attributes

- double x_init
- double y_init
- double myu1
- double myu2
- double lambda1
- double lambda2

5.4.1 Detailed Description

Structure to store Advanced Logistic Map parameters.

Definition at line 151 of file config.hpp.

5.4.2 Member Data Documentation

5.4.2.1 double config::lma::x_init

Definition at line 153 of file config.hpp.

Referenced by pattern::assignMapParameters(), pattern::initializeMapParameters(), and pattern::selectChaotic↩ Map().

5.4.2.2 double config::lma::y_init

Definition at line 154 of file config.hpp.

Referenced by pattern::assignMapParameters(), pattern::initializeMapParameters(), and pattern::selectChaotic↩ Map().

5.4.2.3 double config::lma::myu1

Definition at line 155 of file config.hpp.

Referenced by pattern::assignMapParameters(), and pattern::initializeMapParameters().

5.4.2.4 double config::lma::myu2

Definition at line 156 of file config.hpp.

Referenced by pattern::assignMapParameters(), and pattern::initializeMapParameters().

5.4.2.5 double config::lma::lambda1

Definition at line 157 of file config.hpp.

Referenced by pattern::assignMapParameters(), and pattern::initializeMapParameters().

5.4.2.6 double config::lma::lambda2

Definition at line 158 of file config.hpp.

Referenced by pattern::assignMapParameters(), and pattern::initializeMapParameters().

The documentation for this struct was generated from the following file:

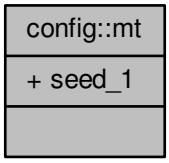
- include/config.hpp

5.5 config::mt Struct Reference

Structure to store Mersenne Twister parameters.

`#include <config.hpp>`

Collaboration diagram for config::mt:



Public Attributes

- int seed_1

5.5.1 Detailed Description

Structure to store Mersenne Twister parameters.

Definition at line 176 of file config.hpp.

5.5.2 Member Data Documentation

5.5.2.1 int config::mt::seed_1

Definition at line 178 of file config.hpp.

Referenced by pattern::assignMapParameters(), and pattern::initializeMapParameters().

The documentation for this struct was generated from the following file:

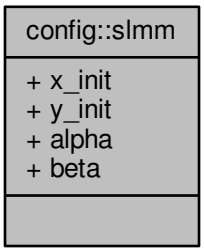
- include/config.hpp

5.6 config::slmm Struct Reference

Structure to store Sine Logistic Modulation Map parameters.

`#include <config.hpp>`

Collaboration diagram for config::slmm:



Public Attributes

- double x_init
- double y_init
- double alpha
- double beta

5.6.1 Detailed Description

Structure to store Sine Logistic Modulation Map parameters.

Definition at line 139 of file config.hpp.

5.6.2 Member Data Documentation

5.6.2.1 double config::slmm::x_init

Definition at line 141 of file config.hpp.

Referenced by pattern::assignMapParameters(), pattern::initializeMapParameters(), and pattern::selectChaotic↩ Map().

5.6.2.2 double config::slmm::y_init

Definition at line 142 of file config.hpp.

Referenced by pattern::assignMapParameters(), pattern::initializeMapParameters(), and pattern::selectChaotic↩ Map().

5.6.2.3 double config::slmm::alpha

Definition at line 143 of file config.hpp.

Referenced by pattern::assignMapParameters(), and pattern::initializeMapParameters().

5.6.2.4 double config::slmm::beta

Definition at line 144 of file config.hpp.

Referenced by pattern::assignMapParameters(), and pattern::initializeMapParameters().

The documentation for this struct was generated from the following file:

- include/config.hpp

Chapter 6

File Documentation

6.1 decrypt.cpp File Reference

```
#include "include/commonheader.hpp"
#include "include/serial.hpp"
#include "include/pattern.hpp"
#include "include/kernel.hpp"
#include "include/io.hpp"
```

Include dependency graph for decrypt.cpp:

Functions

- int main (int argc, char ∗argv[ ])

6.1.1 Function Documentation

6.1.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Get encrypted file path

Get encrypted image name from file path

Reading number of rotation rounds

Reading number of swapping rounds

Reading seed for shuffling map_array

Generating and swapping chaotic map lut

Chaotic map choice variables

Parameter arrays

Assigning chaotic map choices for each vector

Reading map parameters

Diplaying map parameters

CPU vector declarations

GPU Vector Declarations

Allocating GPU memory

Flattening 2D N X M image to 1D N X M vector

Warming up kernel

Undiffusion

Row and Column Unswapping

Display map parameters

Vector generation for row and coulumn swapping

Transferring CPU vectors to GPU memory

Getting results from GPU memory

Swapping

Row and Column Unrotation

Display map parameters

Vector generation for row and column unrotation

Transferring CPU Vectors to GPU Memory

Getting results from GPU

Swapping

Comparison of plain and decrypted images

Compare SHA256 hashes of plain image and decrypted image

Definition at line 7 of file decrypt.cpp.

References common::calc_sha256(), common::checkImages(), config::constant_parameters_file_path, DEBUG_↩ INTERMEDIATE_IMAGES, DEBUG_MAP_CHOICES_ARRAY, DEBUG_MAP_PARAMETERS, DEBUG_READ↩ _WRITE, DEBUG_VECTORS, DIFFUSION, pattern::displayMapParameters(), common::flattenImage(), common↩ ::genMapLUTVec(), common::getFileNameFromPath(), serial::grayLevelTransform(), pattern::MTSequence(), common::printArray32(), common::printArray8(), ROW_COL_ROTATION, ROW_COL_SWAPPING, run_Dec↩ GenCatMap(), run_decRowColSwap(), run_WarmUp(), pattern::rwMapParameters(), pattern::selectChaoticMap(), and common::swapLUT().

```
8 {
12   std::string input_image_path = std::string(argv[1]);
13
14   if(input_image_path == "")
15   {
16     cout<<"\nNo image name specified.Please specify an image name.\nExiting...";
17     exit(0);
18   }
19
23   std::string image_name = common::getFileNameFromPath(input_image_path);
24
25   input_image_path = "";
26   input_image_path = image_name + "_diffused" + ".png";
27
28   auto start = std::chrono::system_clock::now();
29
30   cv::Mat image;
31   image = cv::imread(input_image_path,cv::IMREAD_UNCHANGED);
32
33   if(!image.data)
34   {
35     cout<<"\nCould not open image from "<<input_image_path<<" \nExiting...";
36     exit(0);
37   }
38
39
40   uint32_t m = 0,n = 0,channels = 0, total = 0;
41   double alpha = 0.00,beta = 0.00;
42   int number_of_rotation_rounds = 0, number_of_swapping_rounds = 0,i = 0;
43   long ptr_position = 0;
44   m = image.rows;
45   n = image.cols;
46   channels = image.channels();
47   total = m * n;
48   int fread_status = 9,fseek_status = 9,seed = 0;
49
50   cout<<"\nRows = "<<m;
51   cout<<"\nCols = "<<n;
52   cout<<"\nChannels = "<<channels;
53   uint32_t *map_choice_array = (uint32_t*)calloc(6,sizeof(uint32_t));
54   uint32_t *map_array = (uint32_t*)calloc(15,sizeof(uint32_t));
55
59   FILE *infile = fopen(config::constant_parameters_file_path,"rb");
60
61   if(infile == NULL)
62   {
63     cout<<"\nCould not open "<<config::constant_parameters_file_path<<
      " for reading\nExiting...";
64     exit(0);
65   }
66
67   if(DEBUG_READ_WRITE == 1)
68   {
69     cout<<"\npointer position before reading the number of rotation rounds = "<<ptr_position;
70   }
71
72   fread_status = fread(&number_of_rotation_rounds,sizeof(number_of_rotation_rounds),1,infile);
73   ptr_position =  ftell(infile);
74
75   if(DEBUG_READ_WRITE == 1)
76   {
77     cout<<"\npointer position after reading the number of rotation rounds = "<<ptr_position;
78   }
79
80   fclose(infile);
81
85   infile = fopen(config::constant_parameters_file_path,"rb");
86
87   if(infile == NULL)
88   {
89     cout<<"\nCould not open "<<config::constant_parameters_file_path<<
      " for reading\nExiting...";
90     exit(0);
91   }
92
93   if(ptr_position > 0)
94   {
95      fseek_status = fseek(infile,(ptr_position),SEEK_SET);
96      ptr_position = ftell(infile);
97   }
98
99   if(DEBUG_READ_WRITE == 1)
100   {
101     cout<<"\npointer position before reading the number of swapping rounds = "<<ptr_position;
102   }
103
104    fread_status = fread(&number_of_swapping_rounds,sizeof(number_of_swapping_rounds),1,infile);
```

```
105   ptr_position =  ftell(infile);
106
107   if(DEBUG_READ_WRITE == 1)
108   {
109     cout<<"\nfread status after reading the number of swapping rounds = "<<fread_status;
110     cout<<"\npointer position after reading the number of swapping rounds = "<<ptr_position;
111   }
112
113   fclose(infile);
114
119   infile = fopen(config::constant_parameters_file_path,"rb");
120
121   if(infile == NULL)
122   {
123     cout<<"\nCould not open "<<config::constant_parameters_file_path<<
      " for writing\nExiting...";
124     exit(0);
125   }
126
127
128   if(ptr_position > 0)
129   {
130      fseek_status = fseek(infile,(ptr_position),SEEK_SET);
131      ptr_position = ftell(infile);
132   }
133
134   if(DEBUG_READ_WRITE == 1)
135   {
136     cout<<"\npointer position before reading seed = "<<ptr_position;
137   }
138
139   fread_status = fread(&seed,sizeof(seed),1,infile);
140   ptr_position =  ftell(infile);
141
142   if(DEBUG_READ_WRITE == 1)
143   {
144     cout<<"\nfread status after reading seed = "<<fread_status;
145     cout<<"\npointer position after reading seed = "<<ptr_position;
146   }
147
148   fclose(infile);
149
150   if(DEBUG_MAP_PARAMETERS == 1)
151   {
152     cout<<"\nnumber of rotation rounds = "<<number_of_rotation_rounds;
153     cout<<"\nnumber of swapping rounds = "<<number_of_swapping_rounds;
154     cout<<"\nseed = "<<seed;
155   }
156
160   pattern::MTSequence(map_array,15,1,15,seed);
161   common::genMapLUTVec(map_choice_array,6);
162   common::swapLUT(map_choice_array,map_array,6);
163
167   config::ChaoticMap map_row_random_vec;
168   config::ChaoticMap map_col_random_vec;
169   config::ChaoticMap map_row_rotation_vec;
170   config::ChaoticMap map_col_rotation_vec;
171   config::ChaoticMap map_diffusion_array;
172
176   config::lm lm_parameters[6];
177   config::lma lma_parameters[6];
178   config::slmm slmm_parameters[6];
179   config::lasm lasm_parameters[6];
180   config::lalm lalm_parameters[6];
181   config::mt mt_parameters[6];
182
186   map_row_random_vec = config::ChaoticMap(map_choice_array[0]);
187   map_col_random_vec = config::ChaoticMap(map_choice_array[1]);
188   map_row_rotation_vec = config::ChaoticMap(map_choice_array[2]);
189   map_col_rotation_vec = config::ChaoticMap(map_choice_array[3]);
190   map_diffusion_array = config::ChaoticMap(map_choice_array[4]);
191
192
193   if(DEBUG_MAP_CHOICES_ARRAY == 1)
194   {
195     cout<<"\nMap choice array after reading = ";
196     for(int i = 0; i < 6; ++i)
197     {
198       printf(" %d",map_choice_array[i]);
199     }
200   }
201
205   ptr_position = pattern::rwMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_row_rotation_vec,infile,"rb",ptr_position,
      number_of_rotation_rounds);
206   ptr_position = pattern::rwMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_col_rotation_vec,infile,"rb",ptr_position,
```

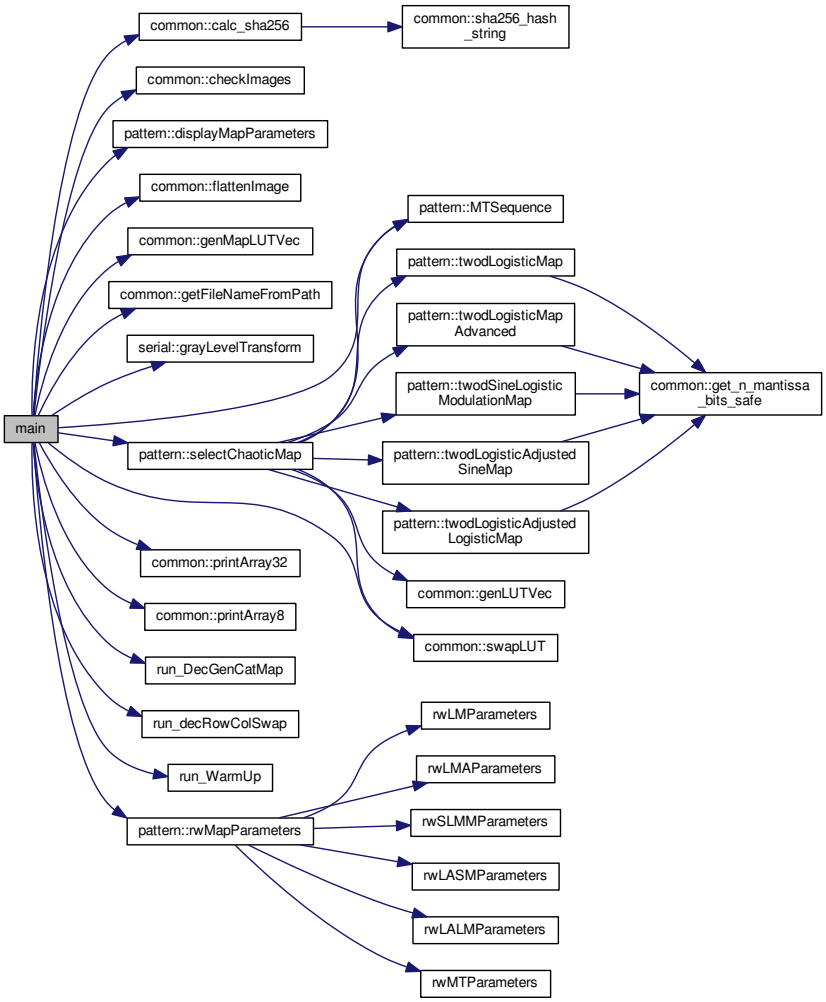
```
    number_of_rotation_rounds);
207 ptr_position = pattern::rwMapParameters(lm_parameters,lma_parameters,
    slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_row_random_vec,infile,"rb",ptr_position,
    number_of_swapping_rounds);
208 ptr_position = pattern::rwMapParameters(lm_parameters,lma_parameters,
    slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_col_random_vec,infile,"rb",ptr_position,
    number_of_swapping_rounds);
209 ptr_position = pattern::rwMapParameters(lm_parameters,lma_parameters,
    slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_diffusion_array,infile,"rb",ptr_position,1);
210
211 if(DEBUG_MAP_PARAMETERS == 1)
212 {
216   pattern::displayMapParameters(lm_parameters,lma_parameters,slmm_parameters
    ,lasm_parameters,lalm_parameters,mt_parameters,map_row_rotation_vec,number_of_rotation_rounds);
217   pattern::displayMapParameters(lm_parameters,lma_parameters,slmm_parameters
    ,lasm_parameters,lalm_parameters,mt_parameters,map_col_rotation_vec,number_of_rotation_rounds);
218   pattern::displayMapParameters(lm_parameters,lma_parameters,slmm_parameters
    ,lasm_parameters,lalm_parameters,mt_parameters,map_row_random_vec,number_of_swapping_rounds);
219   pattern::displayMapParameters(lm_parameters,lma_parameters,slmm_parameters
    ,lasm_parameters,lalm_parameters,mt_parameters,map_col_random_vec,number_of_swapping_rounds);
220   pattern::displayMapParameters(lm_parameters,lma_parameters,slmm_parameters
    ,lasm_parameters,lalm_parameters,mt_parameters,map_diffusion_array,1);
221 }
222
226 uint8_t *enc_vec = (uint8_t*)calloc(total * channels,sizeof(uint8_t));
227 uint8_t *dec_vec = (uint8_t*)calloc(total * channels,sizeof(uint8_t));
228 uint8_t *final_vec = (uint8_t*)calloc(total * channels,sizeof(uint8_t));
229 uint32_t *row_swap_lut_vec = (uint32_t*)calloc(m,sizeof(uint32_t));
230 uint32_t *col_swap_lut_vec = (uint32_t*)calloc(n,sizeof(uint32_t));
231 uint32_t *U = (uint32_t*)calloc(m,sizeof(uint32_t));
232 uint32_t *V = (uint32_t*)calloc(n,sizeof(uint32_t));
233 double *x = (double*)calloc(total * channels,sizeof(double));
234 double *y = (double*)calloc(total * channels,sizeof(double));
235 double *x_bar = (double*)calloc(total * channels,sizeof(double));
236 double *y_bar = (double*)calloc(total * channels,sizeof(double));
237 uint32_t *row_rotation_vec = (uint32_t*)calloc(total * channels,sizeof(uint32_t));
238 uint32_t *col_rotation_vec = (uint32_t*)calloc(total * channels,sizeof(uint32_t));
239 uint32_t *row_random_vec = (uint32_t*)calloc(total * channels,sizeof(uint32_t));
240 uint32_t *col_random_vec = (uint32_t*)calloc(total * channels,sizeof(uint32_t));
241
242
243 uint32_t *diffusion_array = (uint32_t*)calloc(total * channels,sizeof(uint32_t));
244 uint32_t *dummy_lut_vec = (uint32_t*)calloc(2,sizeof(uint32_t));
245
249 uint8_t *gpu_enc_vec;
250 uint8_t *gpu_dec_vec;
251 uint8_t *gpu_final_vec;
252 const uint32_t *gpu_row_swap_lut_vec;
253 const uint32_t *gpu_col_swap_lut_vec;
254 const uint32_t *gpu_U;
255 const uint32_t *gpu_V;
256 const uint32_t *gpu_diffusion_array;
257
261 cudaMalloc((void**)&gpu_enc_vec,total * channels * sizeof(uint8_t));
262 cudaMalloc((void**)&gpu_dec_vec,total * channels * sizeof(uint8_t));
263 cudaMalloc((void**)&gpu_final_vec,total * channels * sizeof(uint8_t));
264 cudaMalloc((void**)&gpu_row_swap_lut_vec,m * sizeof(uint32_t));
265 cudaMalloc((void**)&gpu_col_swap_lut_vec,n * sizeof(uint32_t));
266 cudaMalloc((void**)&gpu_U,m * sizeof(uint32_t));
267 cudaMalloc((void**)&gpu_V,n * sizeof(uint32_t));
268 cudaMalloc((void**)&gpu_diffusion_array,total * channels * sizeof(uint32_t));
269
270
274 common::flattenImage(image,enc_vec,channels);
275
276 if(DEBUG_VECTORS == 1)
277 {
278   cout<<"\nencrypted image = ";
279   common::printArray8(enc_vec,total * channels);
280 }
281
285 dim3 warm_up_grid(1,1,1);
286 dim3 warm_up_block(1,1,1);
287 run_WarmUp(warm_up_grid,warm_up_block);
288
292 if(DIFFUSION == 1)
293 {
294   pattern::selectChaoticMap(lm_parameters,lma_parameters,slmm_parameters,
    lasm_parameters,lalm_parameters,mt_parameters,x,y,x_bar,y_bar,diffusion_array,dummy_lut_vec,map_diffusion_array,0,2
    ,total * channels);
295
296   auto undiffusion_start = std::chrono::system_clock::now();
297   serial::grayLevelTransform(enc_vec,diffusion_array,total * channels);
298
299   auto undiffusion_end = std::chrono::system_clock::now();
300
301   auto undiffusion_time = std::chrono::duration_cast<std::chrono::milliseconds>(undiffusion_end -
```

```
    undiffusion_start).count();
302   cout<<"\nUndiffusion time =  "<<undiffusion_time<<" ms";
303 
304   if(DEBUG_VECTORS == 1)
305   {
306     printf("\n\nUndiffused image = ");
307     common::printArray8(enc_vec,total * channels);
308   }
309 
310   if(DEBUG_INTERMEDIATE_IMAGES == 1)
311   {
312     std::string undiffused_image = "";
313     undiffused_image = image_name + "_undiffused_" + ".png";
314     cv::Mat img_reshape(m,n,image.type(),enc_vec);
315     bool undiffusion_status = cv::imwrite(undiffused_image,img_reshape);
316 
317     if(undiffusion_status == 1)
318     {
319       cout<<"\nUNDIFFUSION SUCCESSFUL";
320     }
321 
322     else
323     {
324       cout<<"\nUNDIFFUSION UNSUCCESSFUL";
325     }
326   }
327 }
328 
332 if(ROW_COL_SWAPPING == 1)
333 {
337   for(i = number_of_swapping_rounds - 1; i >= 0; --i)
338   {
339 
343     pattern::selectChaoticMap(lm_parameters,lma_parameters,slmm_parameters,
    lasm_parameters,lalm_parameters,mt_parameters,x,y,x_bar,y_bar,row_random_vec,row_swap_lut_vec,map_row_random_vec,
    i,m,total * channels);
344     pattern::selectChaoticMap(lm_parameters,lma_parameters,slmm_parameters,
    lasm_parameters,lalm_parameters,mt_parameters,x,y,x_bar,y_bar,col_random_vec,col_swap_lut_vec,map_col_random_vec,
    i,n,total * channels);
345 
346     dim3 dec_row_col_swap_grid(m,n,1);
347     dim3 dec_row_col_swap_blocks(channels,1,1);
348 
352     cudaMemcpy(gpu_enc_vec,enc_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
353     cudaMemcpy(gpu_dec_vec,dec_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
354     cudaMemcpy((void*)gpu_row_swap_lut_vec,row_swap_lut_vec,m * sizeof(uint32_t),cudaMemcpyHostToDevice);
355     cudaMemcpy((void*)gpu_col_swap_lut_vec,col_swap_lut_vec,n * sizeof(uint32_t),cudaMemcpyHostToDevice);
356 
357     run_decRowColSwap(gpu_enc_vec,gpu_dec_vec,(const uint32_t*)gpu_row_swap_lut_vec,(
    const uint32_t*)gpu_col_swap_lut_vec,dec_row_col_swap_grid,dec_row_col_swap_blocks);
358 
362     cudaMemcpy(enc_vec,gpu_enc_vec,total * channels * sizeof(uint8_t),cudaMemcpyDeviceToHost);
363     cudaMemcpy(dec_vec,gpu_dec_vec,total * channels * sizeof(uint8_t),cudaMemcpyDeviceToHost);
364 
368     if(number_of_swapping_rounds > 1)
369     {
370       cudaMemcpy(enc_vec,dec_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToHost);
371       cudaMemcpy(gpu_enc_vec,enc_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
372     }
373 
374     if(DEBUG_VECTORS == 1)
375     {
376 
377       cout<<"\n\ni = "<<i;
378       printf("\nenc_vec = ");
379 
380       common::printArray8(enc_vec,total * channels);
381       printf("\ndec_vec = ");
382 
383       common::printArray8(dec_vec,total * channels);
384       cout<<"\nROUND = "<<i;
385 
386       printf("\nrow_swap_lut_vec = ");
387       common::printArray32(row_swap_lut_vec,m);
388 
389       printf("\ncol_swap_lut_vec = ");
390       common::printArray32(col_swap_lut_vec,n);
391     }
392 
393     if(DEBUG_INTERMEDIATE_IMAGES == 1)
394     {
395       std::string unswapped_image = "";
396       unswapped_image = image_name + "_unswapped" + "_ROUND_" + std::to_string(i + 1) + ".png";
397       cv::Mat img_reshape(m,n,image.type(),dec_vec);
398       bool swapping_status = cv::imwrite(unswapped_image,img_reshape);
399 
400       if(swapping_status == 1)
```

```
401         {
402           cout<<"\nUNSWAPPING SUCCESSFUL FOR ROUND "<<i + 1;
403         }
404 
405         else
406         {
407           cout<<"UNSWAPPING UNSUCCESSFUL FOR ROUND "<<i + 1;
408         }
409       }
410    }
411 
412 
413   }
414 
419   if(ROW_COL_ROTATION == 1)
420   {
421 
425     for(i = number_of_rotation_rounds - 1; i >= 0; --i)
426     {
427 
428       cout<<"\nROUND "<<i;
429 
433       pattern::selectChaoticMap(lm_parameters,lma_parameters,slmm_parameters,
    lasm_parameters,lalm_parameters,mt_parameters,x,y,x_bar,y_bar,row_rotation_vec,U,map_row_rotation_vec,i,m,total *
     channels);
434       pattern::selectChaoticMap(lm_parameters,lma_parameters,slmm_parameters,
    lasm_parameters,lalm_parameters,mt_parameters,x,y,x_bar,y_bar,col_rotation_vec,V,map_col_rotation_vec,i,n,total *
     channels);
435 
436       dim3 dec_gen_cat_map_grid(m,n,1);
437       dim3 dec_gen_cat_map_blocks(channels,1,1);
438 
442       cudaMemcpy(gpu_dec_vec,dec_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
443       cudaMemcpy(gpu_final_vec,final_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
444       cudaMemcpy((void*)gpu_U,U,m * sizeof(uint32_t),cudaMemcpyHostToDevice);
445       cudaMemcpy((void*)gpu_V,V,n * sizeof(uint32_t),cudaMemcpyHostToDevice);
446 
447       run_DecGenCatMap(gpu_dec_vec,gpu_final_vec,(const uint32_t*)gpu_V,(const uint32_t*)
    gpu_U,dec_gen_cat_map_grid,dec_gen_cat_map_blocks);
448 
452       cudaMemcpy(dec_vec,gpu_dec_vec,total * channels * sizeof(uint8_t),cudaMemcpyDeviceToHost);
453       cudaMemcpy(final_vec,gpu_final_vec,total * channels * sizeof(uint8_t),cudaMemcpyDeviceToHost);
454 
458       if(number_of_rotation_rounds > 1)
459       {
460         cudaMemcpy(dec_vec,final_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToHost);
461         cudaMemcpy(gpu_dec_vec,dec_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
462       }
463 
464       if(DEBUG_VECTORS == 1)
465       {
466 
467         cout<<"\ni = "<<i;
468 
469         printf("\ndec_vec = ");
470         common::printArray8(dec_vec,total * channels);
471 
472         printf("\nfinal_vec = ");
473         common::printArray8(final_vec,total * channels);
474 
475         cout<<"\nU = ";
476         common::printArray32(U,m);
477 
478         cout<<"\nV = ";
479         common::printArray32(V,n);
480 
481         cout<<"\nRow rotation vec = ";
482         common::printArray32(row_rotation_vec,total * channels);
483 
484         cout<<"\nColumn rotation vec = ";
485         common::printArray32(col_rotation_vec,total * channels);
486 
487       }
488 
489       if(DEBUG_INTERMEDIATE_IMAGES == 1)
490       {
491 
492         std::string unrotated_image = "";
493         unrotated_image = image_name + "_unrotated" + "_ROUND_" + std::to_string(i + 1) + ".png";
494         cv::Mat img_reshape(m,n,image.type(),final_vec);
495         bool unrotation_status = cv::imwrite(unrotated_image,img_reshape);
496 
497         if(unrotation_status == 1)
498         {
499           cout<<"\nUNROTATION SUCCESSFUL FOR ROUND "<<i + 1;
500         }
501 
```

```
502         else
503         {
504           cout<<"\nUNROTATION UNSUCCESSFUL FOR ROUND "<<i + 1;
505         }
506
507       }
508
509
510     }
511   }
512
513   auto end = std::chrono::system_clock::now();
514
515   auto elapsed = std::chrono::duration_cast<std::chrono::microseconds>(end - start).count();
516   cout<<"\nTotal decryption time = "<<elapsed / 1000000<<" s";
517
518
519   std::string plain_image_path = std::string(argv[1]);
520   std::string decrypted_image_path = image_name + "_unrotated" + "_ROUND_1" + ".png";
521
522   char *constant_plain_image_path = const_cast<char*>(plain_image_path.c_str());
523   char *constant_decrypted_image_path = const_cast<char*>(decrypted_image_path.c_str());
524
528   cv::Mat plain_image = cv::imread(plain_image_path,cv::IMREAD_UNCHANGED);
529   cv::Mat decrypted_image = cv::imread(decrypted_image_path,cv::IMREAD_UNCHANGED);
530   bool decryption_status = common::checkImages(plain_image,decrypted_image);
531
532
536   std::string plain_image_hash = common::calc_sha256(constant_plain_image_path);
537   std::string decrypted_image_hash = common::calc_sha256(constant_decrypted_image_path);
538   bool hash_equality_status = 0;
539
540   if(plain_image_hash == decrypted_image_hash)
541   {
542     hash_equality_status = 1;
543   }
544
545   else
546   {
547     hash_equality_status = 0;
548   }
549
550
551   if(decryption_status == 0)
552   {
553     cout<<"\nDECRYPTION UNSUCCESSFUL\nHASHES DON'T MATCH AND DIFFERENCE IMAGE IS NONZERO";
554   }
555
556   else if(decryption_status == 1 and hash_equality_status == 1)
557   {
558     cout<<"\nDECRYPTION SUCCESSFUL\nHASHES EQUAL AS IMAGES HAVE BEEN MODIFIED THROUGH OPENCV";
559   }
560
561   else if(decryption_status == 1 and hash_equality_status == 0)
562   {
563      cout<<"\nDECRYPTION SUCCESSFUL\nHASHES UNEQUAL AS IMAGES HAVE NOT BEEN MODIFIED THROUGH OPENCV";
564   }
565
566   else
567   {
568     cout<<"\nDECRYPTION UNSUCCESSFUL\nHASHES DON'T MATCH AND DIFFERENCE IMAGE IS NONZERO";
569   }
570
571
572   return 0;
573 }
```

Here is the call graph for this function:

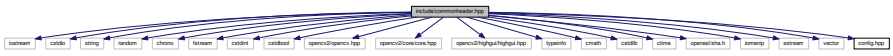


6.2 include/commonheader.hpp File Reference

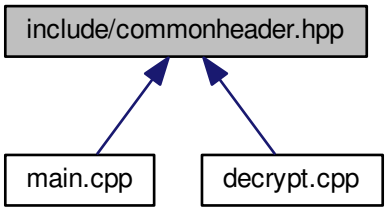
```
#include <iostream>
```

```
#include <cstdio>
#include <string>
#include <random>
#include <chrono>
#include <fstream>
#include <cstdint>
#include <cstdbool>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <typeinfo>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <openssl/sha.h>
#include <iomanip>
#include <sstream>
#include <vector>
#include "config.hpp"
```

Include dependency graph for commonheader.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- common

Functions

- static void common::flattenImage (cv::Mat image, uint8_t *&img_vec, uint32_t channels)

  Converts an image of dimensions N x M into a 1D vector of length N x M.
- static void common::printImageContents (cv::Mat image, uint32_t channels)

  Prints the gray level values in a cv::Mat image in row major order.
- static uint8_t common::checkOverflow (uint16_t number_1, uint16_t number_2)

Checks if the product of 2 16-bit unsigned integers exceeds 255.

- static void common::show_ieee754 (double f)

  formatted output of ieee-754 representation of double-precision floating-point
- static void common::print_int_bits (int num)

  Print bits of a 32-bit signed integer.
- static uint16_t common::get_n_mantissa_bits_safe (double f, int number_of_bits)

  Transfers the last n bits from a double to an n-bit unsigned integer.
- static void common::writeVectorToFile32 (uint32_t *&vec, int length, std::string filename)

  Writes a 32-bit vector to a .txt file.
- static void common::writeVectorToFile8 (uint8_t *&vec, int length, std::string filename)

  Writes an 8-bit unsigned integer vector to a .txt file.
- static void common::printArray8 (uint8_t *&arr, int length)

  Prints an 8-bit unsigned integer array of length 'length'.
- static void common::printArray16 (uint16_t *&arr, int length)

  Prints a 16-bit unsigned integer array of length 'length'.
- static void common::printArray32 (uint32_t *&arr, int length)

  Prints a 32-bit unsigned integer array of length 'length'.
- static void common::printArrayDouble (double *&arr, int length)

  Prints a double array of length 'length'.
- static uint32_t common::getRandomUnsignedInteger32 (uint32_t lower_bound, uint32_t upper_bound)

  Returns a random 32-bit unsigned integer within a range of (lower_bound,upper_bound).
- static int common::getRandomInteger (int lower_bound, int upper_bound)

  Returns a random 32-bit signed integer within a range of (lower_bound,upper_bound).
- static uint8_t common::getRandomUnsignedInteger8 (uint8_t lower_bound, uint8_t upper_bound)

  Returns a random 8-bit unsigned integer within a range of (lower_bound,upper_bound).
- static double common::getRandomDouble (double lower_limit, double upper_limit)

  Returns a random double within a range of (lower_bound,upper_bound).
- static config::ChaoticMap common::mapAssigner (int lower_limit, int upper_limit)

  Returns a value of type ChaoticMap within a range of (lower_limit,upper_limit).
- static void common::rowColLUTGen (uint32_t *&rowSwapLUT, uint32_t *&rowRandVec, uint32_t *&colSwapLUT, uint32_t *&colRandVec, uint32_t m, uint32_t n)

  Generates shuffled row and column Lookup Tables using Fisher - Yates Shuffle for row and column rotation or swapping.
- static void common::swapLUT (uint32_t *&swapLUT, uint32_t *randVec, uint32_t m)

  Shuffles the Lookup Table used to shuffle chaotic map choices array.
- static void common::genLUTVec (uint32_t *&lut_vec, uint32_t n)

  Generates a Lookup Table with values from 0 to n - 1 in ascending order for row and column swapping or rotating.
- static void common::genMapLUTVec (uint32_t *&lut_vec, uint32_t n)

  Generates a Lookup Table with values from 1 to n in ascending order for shuffling the chaotic map choices array.
- static std::string common::getFileNameFromPath (std::string filename)

  Gets the file name from the given file path.
- static std::string common::sha256_hash_string (unsigned char hash[SHA256_DIGEST_LENGTH])

  Converts an 8-bit unsigned char SHA256 array into a SHA256 std::string.
- static std::string common::calc_sha256 (const char *path)

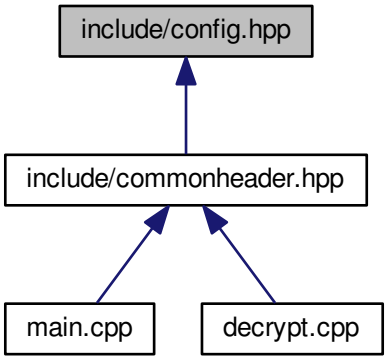
  Calculates SHA256 Hash of a given file.
- static void common::checkImageVectors (uint8_t *plain_img_vec, uint8_t *decrypted_img_vec, uint32_t total)

  Finds differences between two image vectors.
- static bool common::checkImages (cv::Mat image_1, cv::Mat image_2)

  Finds differences between two 2D N x M images.

6.3 include/config.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct config::lalm

  Structure to store Logistic Adjusted Logistic Map parameters.
- struct config::lasm

  Structure to store Logistic Adjusted Sine Map parameters.
- struct config::slmm

  Structure to store Sine Logistic Modulation Map parameters.
- struct config::lma

  Structure to store Advanced Logistic Map parameters.
- struct config::lm

  Structure to store Logistic Map parameters.
- struct config::mt

  Structure to store Mersenne Twister parameters.

Namespaces

- config

Macros

- #define DEBUG_READ_WRITE 0

  All control flags.
- #define DEBUG_VECTORS 0
- #define DEBUG_IMAGES 1
- #define DEBUG_INTERMEDIATE_IMAGES 1
- #define DEBUG_MAP_PARAMETERS 0

- #define DEBUG_MAP_CHOICES_ARRAY 0
- #define ROW_COL_SWAPPING 1
- #define ROW_COL_ROTATION 1
- #define DIFFUSION 1
- #define ROUNDS_LOWER_LIMIT 1
- #define ROUNDS_UPPER_LIMIT 5
- #define X_LOWER_LIMIT 0.10000000
- #define X_UPPER_LIMIT 0.20000000
- #define Y_LOWER_LIMIT 0.10000000
- #define Y_UPPER_LIMIT 0.20000000
- #define MYU_LOWER_LIMIT 0.50000000
- #define MYU_UPPER_LIMIT 0.90000000
- #define LASM_LOWER_LIMIT 0.40000000
- #define LASM_UPPER_LIMIT 0.90000000
- #define MYU1_LOWER_LIMIT 3.01000000
- #define MYU1_UPPER_LIMIT 3.29000000
- #define MYU2_LOWER_LIMIT 3.01000000
- #define MYU2_UPPER_LIMIT 3.30000000
- #define LAMBDA1_LOWER_LIMIT 0.16000000
- #define LAMBDA1_UPPER_LIMIT 0.21000000
- #define LAMBDA2_LOWER_LIMIT 0.14000000
- #define LAMBDA2_UPPER_LIMIT 0.15000000
- #define ALPHA_LOWER_LIMIT 0.90500000
- #define ALPHA_UPPER_LIMIT 1.00000000
- #define BETA_LOWER_LIMIT 2.97000000
- #define BETA_UPPER_LIMIT 3.20000000
- #define R_LOWER_LIMIT 1.11000000
- #define R_UPPER_LIMIT 1.19000000
- #define SEED_LOWER_LIMIT 30000
- #define SEED_UPPER_LIMIT 90000
- #define MAP_LOWER_LIMIT 1
- #define MAP_UPPER_LIMIT 5
- #define LOWER_LIMIT 0.000001
- #define UPPER_LIMIT 0.09
- #define NUMBER_OF_BITS 31
- #define INIT 100
- #define BIT_RETURN(A, LOC) (((A >> LOC) & 0x1) ? 1:0)

Enumerations

- enum config::ChaoticMap {
  config::ChaoticMap::TwoDLogisticMap = 1, config::ChaoticMap::TwoDLogisticMapAdvanced, config::↩
  ChaoticMap::TwoDLogisticAdjustedSineMap, config::ChaoticMap::TwoDSineLogisticModulationMap,
  config::ChaoticMap::TwoDLogisticAdjustedLogisticMap }

  Enumeration class used to denote Chaotic Map choices.

Variables

- int config::lower_limit = 1

  Range of pseudorandom values to be produced by Mersenne Twister.
- int config::upper_limit = (rows ∗ cols ∗ 3) + 1
- std::string config::parameters_file = "parameters"

  The file path of the parameter file where all parameters are stored.
- std::string config::binary_extension = ".bin"
- std::string config::parameters_file_path = parameters_file + binary_extension
- char ∗ config::constant_parameters_file_path = const_cast<char∗>(parameters_file_path.c_str())

  char∗ constant strings for use with standard C file handling functions

6.3.1 Macro Definition Documentation

6.3.1.1 #define DEBUG_READ_WRITE 0

All control flags.

Use this header file to configure operation of the algorithm

Definition at line 34 of file config.hpp.

Referenced by main(), rwLALMParameters(), rwLASMParameters(), rwLMAParameters(), rwLMParameters(), rwMTParameters(), and rwSLMMParameters().

6.3.1.2 #define DEBUG_VECTORS 0

Definition at line 35 of file config.hpp.

Referenced by main().

6.3.1.3 #define DEBUG_IMAGES 1

Definition at line 37 of file config.hpp.

6.3.1.4 #define DEBUG_INTERMEDIATE_IMAGES 1

Definition at line 38 of file config.hpp.

Referenced by main().

6.3.1.5 #define DEBUG_MAP_PARAMETERS 0

Definition at line 39 of file config.hpp.

Referenced by main().

6.3.1.6 #define DEBUG_MAP_CHOICES_ARRAY 0

Definition at line 40 of file config.hpp.

Referenced by main().

6.3.1.7 #define ROW_COL_SWAPPING 1

Definition at line 42 of file config.hpp.

Referenced by main().

6.3.1.8 #define ROW_COL_ROTATION 1

Definition at line 43 of file config.hpp.

Referenced by main().

6.3.1.9 #define DIFFUSION 1

Definition at line 44 of file config.hpp.

Referenced by main().

6.3.1.10 #define ROUNDS_LOWER_LIMIT 1

Definition at line 46 of file config.hpp.

6.3.1.11 #define ROUNDS_UPPER_LIMIT 5

Definition at line 47 of file config.hpp.

6.3.1.12 #define X_LOWER_LIMIT 0.10000000

Definition at line 49 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.13 #define X_UPPER_LIMIT 0.20000000

Definition at line 50 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.14 #define Y_LOWER_LIMIT 0.10000000

Definition at line 52 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.15 #define Y_UPPER_LIMIT 0.20000000

Definition at line 53 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.16 #define MYU_LOWER_LIMIT 0.50000000

Definition at line 55 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.17 #define MYU_UPPER_LIMIT 0.90000000

Definition at line 56 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.18 #define LASM_LOWER_LIMIT 0.40000000

Definition at line 58 of file config.hpp.

6.3.1.19 #define LASM_UPPER_LIMIT 0.90000000

Definition at line 59 of file config.hpp.

6.3.1.20 #define MYU1_LOWER_LIMIT 3.01000000

Definition at line 61 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.21 #define MYU1_UPPER_LIMIT 3.29000000

Definition at line 62 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.22 #define MYU2_LOWER_LIMIT 3.01000000

Definition at line 64 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.23 #define MYU2_UPPER_LIMIT 3.30000000

Definition at line 65 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.24 #define LAMBDA1_LOWER_LIMIT 0.16000000

Definition at line 67 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.25 #define LAMBDA1_UPPER_LIMIT 0.21000000

Definition at line 68 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.26 #define LAMBDA2_LOWER_LIMIT 0.14000000

Definition at line 70 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.27 #define LAMBDA2_UPPER_LIMIT 0.15000000

Definition at line 71 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.28 #define ALPHA_LOWER_LIMIT 0.90500000

Definition at line 73 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.29 #define ALPHA_UPPER_LIMIT 1.00000000

Definition at line 74 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.30 #define BETA_LOWER_LIMIT 2.97000000

Definition at line 76 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.31 #define BETA_UPPER_LIMIT 3.20000000

Definition at line 77 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.32 #define R_LOWER_LIMIT 1.11000000

Definition at line 79 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.33 #define R_UPPER_LIMIT 1.19000000

Definition at line 80 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.34 #define SEED_LOWER_LIMIT 30000

Definition at line 82 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.35 #define SEED_UPPER_LIMIT 90000

Definition at line 83 of file config.hpp.

Referenced by pattern::assignMapParameters().

6.3.1.36 #define MAP_LOWER_LIMIT 1

Definition at line 85 of file config.hpp.

6.3.1.37 #define MAP_UPPER_LIMIT 5

Definition at line 86 of file config.hpp.

6.3.1.38 #define LOWER_LIMIT 0.000001

Definition at line 88 of file config.hpp.

6.3.1.39 #define UPPER_LIMIT 0.09

Definition at line 89 of file config.hpp.

6.3.1.40 #define NUMBER_OF_BITS 31

Definition at line 90 of file config.hpp.

Referenced by pattern::twodLogisticAdjustedLogisticMap(), pattern::twodLogisticAdjustedSineMap(), pattern::twodLogisticMap(), pattern::twodLogisticMapAdvanced(), and pattern::twodSineLogisticModulationMap().

6.3.1.41 #define INIT 100

Definition at line 91 of file config.hpp.

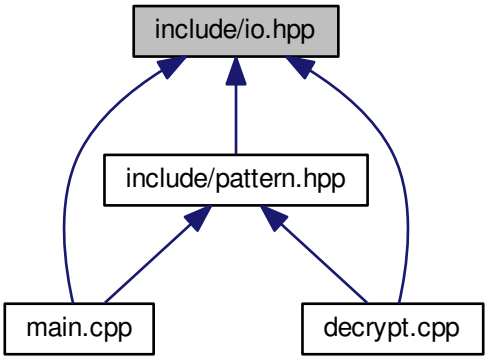
6.3.1.42 #define BIT_RETURN(*A, LOC*) (((A >> LOC) & 0x1) ? 1:0)

Definition at line 92 of file config.hpp.

Referenced by common::get_n_mantissa_bits_safe(), and common::show_ieee754().

6.4 include/io.hpp File Reference

This graph shows which files directly or indirectly include this file:



Functions

- static long rwLMParameters (FILE *outfile, const char *file_path, const char *mode, config::lm lm_↩ parameters[ ], int iteration, int number_of_rounds, long ptr_position)

  This header file contains functions for reading and writing parameters.
- static long rwLMAParameters (FILE *outfile, const char *file_path, const char *mode, config::lma lma_↩ parameters[ ], int iteration, int number_of_rounds, long ptr_position)

  Read or write Logistic Map Advanced parameters.
- static long rwSLMMParameters (FILE *outfile, const char *file_path, const char *mode, config::slmm slmm↩ _parameters[ ], int iteration, int number_of_rounds, long ptr_position)

  Read or write Sine Logistic Modulation Map parameters.
- static long rwLASMParameters (FILE *outfile, const char *file_path, const char *mode, config::lasm lasm_↩ parameters[ ], int iteration, int number_of_rounds, long ptr_position)

  Read or write Logistic Adjusted Sine Map parameters.
- static long rwLALMParameters (FILE *outfile, const char *file_path, const char *mode, config::lalm lalm_↩ parameters[ ], int iteration, int number_of_rounds, long ptr_position)

  Read or write Logistic Adjusted Logistic Map parameters.
- static long rwMTParameters (FILE *outfile, const char *file_path, const char *mode, config::mt mt_↩ parameters[ ], int iteration, int number_of_rounds, long ptr_position)

  Read or write Mersenne Twister seed.

6.4.1 Function Documentation

6.4.1.1 static long rwLMParameters (FILE * *outfile*, const char * *file_path*, const char * *mode*, config::lm *lm_parameters[]*, int *iteration*, int *number_of_rounds*, long *ptr_position*) [inline], [static]

This header file contains functions for reading and writing parameters.

Read or write Logistic Map parameters.

Takes Logistic Map parameters, filepath, iteration and number of rounds as arguments Offset pointer position by length of previous record

Write LM parameters to file

Update pointer position after writing

Ofsetting pointer position by length of revious record

Read LM parameters from file

Update pointer position after reading

Definition at line 19 of file io.hpp.

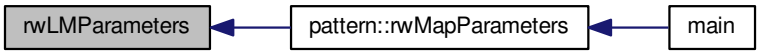
References DEBUG_READ_WRITE.

Referenced by pattern::rwMapParameters().

```
20   {
21     long pointer_position = ptr_position;
22     int fwrite_status = 9,fseek_status = 9,fread_status = 9;
23     outfile = fopen(file_path,mode);
24
25     if(outfile == NULL)
26     {
27       cout<<"\nCould not open "<<file_path<<"\nExiting...";
28       exit(0);
29     }
30
34     if(strcmp("ab",mode) == 0 || strcmp("wb",mode) == 0)
35     {
36       if(pointer_position > 0)
37       {
38         fseek_status = fseek(outfile,(pointer_position + 1),SEEK_SET);
39         pointer_position = ftell(outfile);
40       }
41
42       if(DEBUG_READ_WRITE == 1)
43       {
44         cout<<"\n\niteration = "<<iteration;
45         cout<<"\nfseek status before writing lm_parameters = "<<fseek_status;
46         cout<<"\npointer position before writing lm_parameters = "<<pointer_position;
47       }
48
52       size_t size = number_of_rounds * sizeof(lm_parameters[0]);
53       fwrite_status = fwrite(lm_parameters,size,1,outfile);
54
58       pointer_position = ftell(outfile);
59
60       if(DEBUG_READ_WRITE == 1)
61       {
62         cout<<"\nfwrite status after writing lm_parameters = "<<fwrite_status;
63         cout<<"\npointer position after writing lm_parameters = "<<pointer_position;
64       }
65
66     }
67
68     else
69     {
70       if(DEBUG_READ_WRITE == 1)
71       {
72         cout<<"\npointer position before reading lm_parameters = "<<pointer_position;
```

```
73        }
74
78        if(pointer_position > 0)
79        {
80          fseek_status = fseek(outfile,(pointer_position),SEEK_SET);
81          pointer_position = ftell(outfile);
82        }
83
84        if(DEBUG_READ_WRITE == 1)
85        {
86          cout<<"\n\niteration = "<<iteration;
87          cout<<"\nfseek status before reading lm_parameters = "<<fseek_status;
88          cout<<"\npointer position before writing lm_parameters = "<<pointer_position;
89        }
90
94        size_t size = number_of_rounds * sizeof(lm_parameters[0]);
95        fread_status = fread(lm_parameters,size,1,outfile);
96
100        pointer_position = ftell(outfile);
101
102        if(DEBUG_READ_WRITE == 1)
103        {
104          cout<<"\nfread status after reading lm_parameters = "<<fread_status;
105          cout<<"\npointer position after reading lm_parameters = "<<pointer_position;
106        }
107      }
108
109      fclose(outfile);
110      return pointer_position;
111    }
```

Here is the caller graph for this function:



6.4.1.2 static long rwLMAParameters (FILE * *outfile*, const char * *file_path*, const char * *mode*, config::lma *lma_parameters[ ]*, int *iteration*, int *number_of_rounds*, long *ptr_position*) [inline], [static]

Read or write Logistic Map Advanced parameters.

Takes Logistic Map Advanced parameters, filepath, iteration and number of rounds as arguments Offset pointer position by length of previous record

Write LMA parameters to file

Update pointer position after writing

Offset pointer position by length of previous record

Read LMA parameters from file

Update pointer position after reading

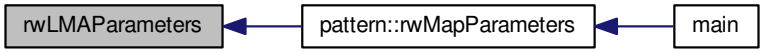
Definition at line 117 of file io.hpp.

References DEBUG_READ_WRITE.

Referenced by pattern::rwMapParameters().

```
118   {
119     long pointer_position = ptr_position;
120     int fwrite_status = 9,fseek_status = 9,fread_status = 9;
121     outfile = fopen(file_path,mode);
122
123     if(outfile == NULL)
124     {
125       cout<<"\nCould not open "<<file_path<<"\nExiting...";
126       exit(0);
127     }
128
129     if(strcmp("wb",mode) == 0 || strcmp("ab",mode) == 0)
130     {
134       if(pointer_position > 0)
135       {
136         fseek_status = fseek(outfile,(pointer_position + 1),SEEK_SET);
137         pointer_position = ftell(outfile);
138       }
139
140       if(DEBUG_READ_WRITE == 1)
141       {
142         cout<<"\n\niteration = "<<iteration;
143         cout<<"\nfseek status before writing lma_parameters = "<<fseek_status;
144         cout<<"\npointer position before writing lma_parameters = "<<pointer_position;
145       }
146
150       size_t size = sizeof(lma_parameters[0]) * number_of_rounds;
151       fwrite_status = fwrite(lma_parameters,size,1,outfile);
152
156       pointer_position = ftell(outfile);
157
158       if(DEBUG_READ_WRITE == 1)
159       {
160         cout<<"\nfwrite status after writing lma_parameters = "<<fwrite_status;
161         cout<<"\npointer position after writing lma_parameters = "<<pointer_position;
162       }
163     }
164
165     else
166     {
167       if(DEBUG_READ_WRITE == 1)
168       {
169         cout<<"\npointer position before reading lma_parameters = "<<pointer_position;
170       }
171
175       if(pointer_position > 0)
176       {
177         fseek_status = fseek(outfile,(pointer_position),SEEK_SET);
178         pointer_position = ftell(outfile);
179       }
180
181       if(DEBUG_READ_WRITE == 1)
182       {
183         cout<<"\n\niteration = "<<iteration;
184         cout<<"\nfseek status before reading lma_parameters = "<<fseek_status;
185         cout<<"\npointer position before reading lma_parameters = "<<pointer_position;
186       }
187
191       size_t size = sizeof(lma_parameters[0]) * number_of_rounds;
192       fread_status = fread(lma_parameters,size,1,outfile);
193
197       pointer_position = ftell(outfile);
198
199       if(DEBUG_READ_WRITE == 1)
200       {
201         cout<<"\nfread status after writing lma_parameters = "<<fread_status;
202         cout<<"\npointer position after reading lma_parameters = "<<pointer_position;
203       }
204     }
205
206     fclose(outfile);
207     return pointer_position;
208   }
```

Here is the caller graph for this function:



6.4.1.3 static long rwSLMMParameters (FILE * *outfile,* const char * *file_path,* const char * *mode,* config::slmm *slmm_parameters[ ],* int *iteration,* int *number_of_rounds,* long *ptr_position*) [inline], [static]

Read or write Sine Logistic Modulation Map parameters.

Takes Sine Logistic Modulation Map parameters, filepath, iteration and number of rounds as arguments Offset pointer position by length of previous record

Write SLMM parameters to file

Update pointer position after writing

Offset pointer position by length of previous record

Read SLMM parameters from file

Update pointer position after reading

Definition at line 214 of file io.hpp.

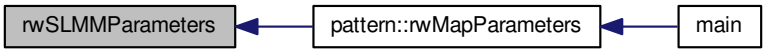
References DEBUG_READ_WRITE.

Referenced by pattern::rwMapParameters().

```
215   {
216     long pointer_position = ptr_position;
217     int fwrite_status = 9,fseek_status = 9,fread_status = 9;
218     outfile = fopen(file_path,mode);
219 
220     if(outfile == NULL)
221     {
222       cout<<"\nCould not open "<<file_path<<"\nExiting...";
223       exit(0);
224     }
225 
226     if(strcmp("wb",mode) == 0 || strcmp("ab",mode) == 0)
227     {
231       if(pointer_position > 0)
232       {
233          fseek_status = fseek(outfile,(pointer_position + 1),SEEK_SET);
234          pointer_position = ftell(outfile);
235       }
236 
237       if(DEBUG_READ_WRITE == 1)
238       {
239         cout<<"\n\niteration = "<<iteration;
240         cout<<"\nfseek status before writing slmm_parameters = "<<fseek_status;
241         cout<<"\npointer position before writing slmm_parameters = "<<pointer_position;
242       }
243 
247       size_t size = sizeof(slmm_parameters[0]) * number_of_rounds;
248       fwrite_status = fwrite(slmm_parameters,size,1,outfile);
249 
253       pointer_position = ftell(outfile);
254       if(DEBUG_READ_WRITE == 1)
```

```
255       {
256         cout<<"\nfwrite status after writing slmm_parameters = "<<fwrite_status;
257         cout<<"\npointer position after writing slmm_parameters = "<<pointer_position;
258       }
259     }
260
261     else
262     {
263       if(DEBUG_READ_WRITE == 1)
264       {
265         cout<<"\npointer position before reading slmm_parameters = "<<pointer_position;
266       }
267
271       if(pointer_position > 0)
272       {
273          fseek_status = fseek(outfile,(pointer_position),SEEK_SET);
274          pointer_position = ftell(outfile);
275       }
276
277       if(DEBUG_READ_WRITE == 1)
278       {
279         cout<<"\n\niteration = "<<iteration;
280         cout<<"\nfseek status before reading slmm_parameters = "<<fseek_status;
281         cout<<"\npointer position before reading slmm_parameters = "<<pointer_position;
282       }
283
287       size_t size = sizeof(slmm_parameters[0]) * number_of_rounds;
288       fread_status = fread(slmm_parameters,size,1,outfile);
289
293       pointer_position = ftell(outfile);
294
295       if(DEBUG_READ_WRITE == 1)
296       {
297         cout<<"\nfread status after reading slmm_parameters = "<<fread_status;
298         cout<<"\npointer position after reading slmm_parameters = "<<pointer_position;
299       }
300     }
301
302     fclose(outfile);
303     return pointer_position;
304   }
```

Here is the caller graph for this function:



6.4.1.4 static long rwLASMParameters (FILE * *outfile*, const char * *file_path*, const char * *mode*, config::lasm *lasm_parameters[ ]*, int *iteration*, int *number_of_rounds*, long *ptr_position*) [inline], [static]

Read or write Logistic Adjusted Sine Map parameters.

Takes Logistic Adjusted Sine Map parameters, filepath, iteration and number of rounds as arguments Offset pointer position by length of previous record

Write LASM parameters to file

Update pointer position after writing

Offset pointer position by length of previous record

Read LASM parameters from file

Update pointer position after reading

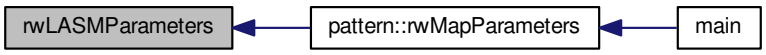
Definition at line 309 of file io.hpp.

References DEBUG_READ_WRITE.

Referenced by pattern::rwMapParameters().

```
310   {
311     long pointer_position = ptr_position;
312     int fwrite_status = 9,fseek_status = 9,fread_status = 9;
313     outfile = fopen(file_path,mode);
314 
315     if(outfile == NULL)
316     {
317       cout<<"\nCould not open "<<file_path<<"\nExiting...";
318       exit(0);
319     }
320 
321     if(strcmp("wb",mode) == 0 || strcmp("ab",mode) == 0)
322     {
326       if(pointer_position > 0)
327       {
328          fseek_status = fseek(outfile,(pointer_position + 1),SEEK_SET);
329          pointer_position = ftell(outfile);
330       }
331 
332       if(DEBUG_READ_WRITE == 1)
333       {
334         cout<<"\n\niteration = "<<iteration;
335         cout<<"\nfseek status before writing lasm_parameters = "<<fseek_status;
336         cout<<"\npointer position before writing lasm_parameters = "<<pointer_position;
337       }
338 
342       size_t size = sizeof(lasm_parameters[0]) * number_of_rounds;
343       fwrite_status = fwrite(lasm_parameters,size,1,outfile);
344 
348       pointer_position = ftell(outfile);
349 
350       if(DEBUG_READ_WRITE == 1)
351       {
352         cout<<"\nfwrite status after writing lasm_parameters = "<<fwrite_status;
353         cout<<"\npointer position after writing lasm_parameters = "<<pointer_position;
354       }
355     }
356 
357     else
358     {
359 
360       if(DEBUG_READ_WRITE == 1)
361       {
362         cout<<"\npointer position before writing lasm_parameters = "<<pointer_position;
363       }
364 
368       if(pointer_position > 0)
369       {
370          fseek_status = fseek(outfile,(pointer_position),SEEK_SET);
371          pointer_position = ftell(outfile);
372       }
373 
374       if(DEBUG_READ_WRITE == 1)
375       {
376         cout<<"\n\niteration = "<<iteration;
377         cout<<"\nfseek status before reading lasm_parameters = "<<fseek_status;
378         cout<<"\npointer position before reading lasm_parameters = "<<pointer_position;
379       }
380 
384       size_t size = sizeof(lasm_parameters[0]) * number_of_rounds;
385       fread_status = fread(lasm_parameters,size,1,outfile);
386 
390       pointer_position = ftell(outfile);
391 
392       if(DEBUG_READ_WRITE == 1)
393       {
394         cout<<"\nfread status after reading lasm_parameters = "<<fread_status;
395         cout<<"\npointer position after reading lasm_parameters = "<<pointer_position;
396       }
397     }
398 
399     fclose(outfile);
400     return pointer_position;
401   }
```

Here is the caller graph for this function:



6.4.1.5 static long rwLALMParameters (FILE * *outfile,* const char * *file_path,* const char * *mode,* config::lalm *lalm_parameters[ ],* int *iteration,* int *number_of_rounds,* long *ptr_position*) [inline], [static]

Read or write Logistic Adjusted Logistic Map parameters.

Takes Logistic Adjusted Logistic Map parameters, filepath, iteration and number of rounds as arguments Offset pointer position by length of previous record

Write LALM parameters to file

Update pointer position after writing

Offset pointer position by length of previous record

Read LALM parameters from file

Update pointer position after reading

Definition at line 406 of file io.hpp.

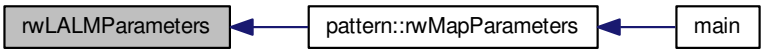
References DEBUG_READ_WRITE.

Referenced by pattern::rwMapParameters().

```
407   {
408     long pointer_position = ptr_position;
409     int fwrite_status = 9,fseek_status = 9,fread_status = 9;
410     outfile = fopen(file_path,mode);
411
412     if(outfile == NULL)
413     {
414       cout<<"\nCould not open "<<file_path<<"\nExiting...";
415       exit(0);
416     }
417
418     if(strcmp("wb",mode) == 0 || strcmp("ab",mode) == 0)
419     {
420
424       if(pointer_position > 0)
425       {
426          fseek_status = fseek(outfile,(pointer_position + 1),SEEK_SET);
427          pointer_position = ftell(outfile);
428       }
429
430       if(DEBUG_READ_WRITE == 1)
431       {
432         cout<<"\n\niteration = "<<iteration;
433         cout<<"\nfseek status before writing lalm_parameters = "<<fseek_status;
434         cout<<"\npointer position before writing lalm_parameters = "<<pointer_position;
435       }
436
440       size_t size = sizeof(lalm_parameters[0]) * number_of_rounds;
441       fwrite_status = fwrite(lalm_parameters,size,1,outfile);
442
446       pointer_position = ftell(outfile);
```

```
447
448       if(DEBUG_READ_WRITE == 1)
449       {
450         cout<<"\nfwrite status after writing lalm_parameters = "<<fwrite_status;
451         cout<<"\npointer position after writing lalm_parameters = "<<pointer_position;
452       }
453     }
454
455     else
456     {
457       if(DEBUG_READ_WRITE == 1)
458       {
459         cout<<"\npointer position before reading lalm_parameters  = "<<pointer_position;
460       }
461
465       if(pointer_position > 0)
466       {
467          fseek_status = fseek(outfile,(pointer_position),SEEK_SET);
468          pointer_position = ftell(outfile);
469       }
470
471       if(DEBUG_READ_WRITE == 1)
472       {
473         cout<<"\n\niteration = "<<iteration;
474         cout<<"\nfseek status before reading lalm_parameters = "<<fseek_status;
475         cout<<"\npointer position before reading lalm_parameters = "<<pointer_position;
476       }
477
481       size_t size = sizeof(lalm_parameters[0]) * number_of_rounds;
482       fread_status = fread(lalm_parameters,size,1,outfile);
483
487       pointer_position = ftell(outfile);
488
489       if(DEBUG_READ_WRITE == 1)
490       {
491         cout<<"\nfread status after writing lalm_parameters = "<<fread_status;
492         cout<<"\npointer position after reading lalm_parameters = "<<pointer_position;
493       }
494     }
495
496     fclose(outfile);
497     return pointer_position;
498   }
```

Here is the caller graph for this function:



6.4.1.6 static long rwMTParameters (FILE * *outfile*, const char * *file_path*, const char * *mode*, config::mt *mt_parameters[ ]*, int *iteration*, int *number_of_rounds*, long *ptr_position*) [inline], [static]

Read or write Mersenne Twister seed.

Takes Mersenne Twister parameters, filepath, iteration and number of rounds as arguments Offset pointer position by length of previous record

Write MT parameters to file

Update pointer position after writing

Offset pointer position by length of previous record

Write MT parameters to file

Update pointer position after writing

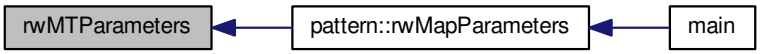
Definition at line 504 of file io.hpp.

References DEBUG_READ_WRITE.

Referenced by pattern::rwMapParameters().

```
505   {
506     int fseek_status = 9,fwrite_status = 9,fread_status = 9;
507     long pointer_position = ptr_position;
508     outfile = fopen(file_path,mode);
509 
510     if(outfile == NULL)
511     {
512       cout<<"\nCould not open "<<file_path<<"\nExiting...";
513       exit(0);
514     }
515 
516     if(strcmp("wb",mode) == 0 || strcmp("ab",mode) == 0)
517     {
521       if(pointer_position > 0)
522       {
523         fseek_status = fseek(outfile,(pointer_position + 1),SEEK_SET);
524         pointer_position = ftell(outfile);
525       }
526 
527       if(DEBUG_READ_WRITE == 1)
528       {
529         cout<<"\n\niteration = "<<iteration;
530         cout<<"\nfseek status before writing basic parameters = "<<fseek_status;
531         cout<<"\npointer position before writing basic parameters = "<<pointer_position;
532       }
533 
537       size_t size = sizeof(mt_parameters[0]) * number_of_rounds;
538       fwrite_status = fwrite(mt_parameters,size,1,outfile);
539 
543       pointer_position = ftell(outfile);
544 
545       if(DEBUG_READ_WRITE == 1)
546       {
547         cout<<"\nfwrite status after writing basic parameters = "<<fwrite_status;
548         cout<<"\npointer position after writing basic parameters = "<<pointer_position;
549       }
550     }
551 
552     else
553     {
554       if(DEBUG_READ_WRITE == 1)
555       {
556         cout<<"\npointer position before reading mt_parameters = "<<pointer_position;
557       }
558 
562       if(pointer_position > 0)
563       {
564         fseek_status = fseek(outfile,(pointer_position),SEEK_SET);
565         pointer_position = ftell(outfile);
566       }
567 
568       if(DEBUG_READ_WRITE == 1)
569       {
570         cout<<"\n\niteration = "<<iteration;
571         cout<<"\nfseek status before reading basic parameters = "<<fseek_status;
572         cout<<"\npointer position before reading basic parameters = "<<pointer_position;
573       }
574 
578       size_t size = sizeof(mt_parameters[0]) * number_of_rounds;
579       fread_status = fread(mt_parameters,size,1,outfile);
583       pointer_position = ftell(outfile);
584 
585       if(DEBUG_READ_WRITE == 1)
586       {
587         cout<<"\nfread status after reading basic parameters = "<<fread_status;
588         cout<<"\npointer position after reading basic parameters = "<<pointer_position;
589       }
590     }
591 
592     fclose(outfile);
593     return pointer_position;
594   }
```

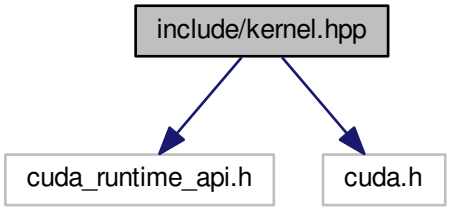
Here is the caller graph for this function:



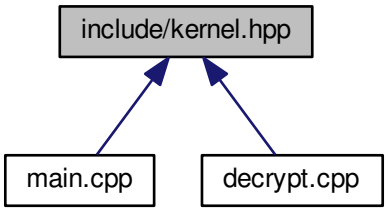
6.5 include/kernel.hpp File Reference

```
#include <cuda_runtime_api.h>
#include <cuda.h>
```

Include dependency graph for kernel.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- void run_WarmUp (dim3 blocks, dim3 block_size)

This header file contains CUDA kernel wrapper function prototypes that are used to interface between the CUDA kernels and the rest of the program.

- void run_EncGenCatMap (uint8_t *in, uint8_t *out, const uint32_t *__restrict__ colRotate, const uint32_t *__restrict__ rowRotate, dim3 blocks, dim3 block_size)

  Rotates image rows and columns.
- void run_DecGenCatMap (uint8_t *in, uint8_t *out, const uint32_t *__restrict__ colRotate, const uint32_t *__restrict__ rowRotate, dim3 blocks, dim3 block_size)

  Unrotates image rows and columns.
- void run_encRowColSwap (uint8_t *img_in, uint8_t *img_out, const uint32_t *__restrict__ rowSwapLUT, const uint32_t *__restrict__ colSwapLUT, dim3 blocks, dim3 block_size)

  Swaps image rows and columns.
- void run_decRowColSwap (uint8_t *img_in, uint8_t *img_out, const uint32_t *__restrict__ rowSwapLUT, const uint32_t *__restrict__ colSwapLUT, dim3 blocks, dim3 block_size)

  Unwaps image rows and columns.

6.5.1 Function Documentation

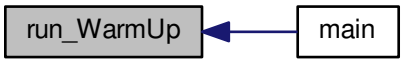
6.5.1.1 void run_WarmUp (dim3 *blocks*, dim3 *block_size*)

This header file contains CUDA kernel wrapper function prototypes that are used to interface between the CUDA kernels and the rest of the program.

Gets GPU ready to perform computation. Helps achieve accurate GPU benchmarking

Referenced by main().

Here is the caller graph for this function:



6.5.1.2 void run_EncGenCatMap (uint8_t * *in*, uint8_t * *out*, const uint32_t *__restrict__ *colRotate*, const uint32_t *__restrict__ *rowRotate*, dim3 *blocks*, dim3 *block_size*)

Rotates image rows and columns.

Based on Arnold Cat Map. Accepts images of dimensions N x N and N x M

Referenced by main().

Here is the caller graph for this function:

6.5.1.3 void run_DecGenCatMap (uint8_t * *in*, uint8_t * *out*, const uint32_t *__restrict__ *colRotate*, const uint32_t *__restrict__ *rowRotate*, dim3 *blocks*, dim3 *block_size*)

Unrotates image rows and columns.

Based on Arnold Cat Map. Accepts images of dimensions N x N and N x M

Referenced by main().

Here is the caller graph for this function:

6.5.1.4 void run_encRowColSwap (uint8_t * *img_in*, uint8_t * *img_out*, const uint32_t *__restrict__ *rowSwapLUT*, const uint32_t *__restrict__ *colSwapLUT*, dim3 *blocks*, dim3 *block_size*)

Swaps image rows and columns.

Accepts images of dimensions N x N and N x M

Referenced by main().

Here is the caller graph for this function:

6.5.1.5 void run_decRowColSwap (uint8_t * *img_in*, uint8_t * *img_out*, const uint32_t *__restrict__ *rowSwapLUT*, const uint32_t *__restrict__ *colSwapLUT*, dim3 *blocks*, dim3 *block_size*)

Unwaps image rows and columns.

Accepts images of dimensions N x N and N x M

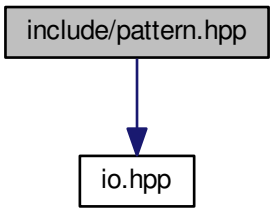
Referenced by main().

Here is the caller graph for this function:

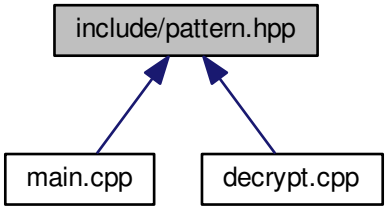
6.6 include/pattern.hpp File Reference

```
#include "io.hpp"
```

Include dependency graph for pattern.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- pattern

  This header file contains chaotic map functions to generate pseudorandom sequences and those functions associated to them.

Functions

- static void pattern::twodLogisticMapAdvanced (double *&x, double *&y, uint32_t *&random_array, double myu1, double myu2, double lambda1, double lambda2, uint32_t number)

  Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLMA.

- static void pattern::twodLogisticAdjustedSineMap (double *&x, double *&y, uint32_t *&random_array, double myu, uint32_t total)

  Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLASM.

- static void pattern::MTSequence (uint32_t *&random_array, uint32_t total, int lower_limit, int upper_limit, int seed)

  Produces a pseudorandom sequence of 32-bit unsigned integers using Mersenne Twister.

- static void pattern::twodSineLogisticModulationMap (double *&x, double *&y, uint32_t *&random_array, double alpha, double beta, uint32_t total)

  Produces a pseudorandom sequence of 32-bit unsigned integers using 2DSLMM.

- static void pattern::twodLogisticAdjustedLogisticMap (double *&x, double *&y, double *&x_bar, double *&y_bar, uint32_t *&random_array, double myu, uint32_t total)

  Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLALM.

- static void pattern::twodLogisticMap (double *&x, double *&y, uint32_t *&random_array, double r, uint32_t total)

  Produces a pseudorandom sequence of 32-bit unsigned integers using 2DLALM.

- static void pattern::initializeMapParameters (config::lm lm_parameters[ ], config::lma lma_parameters[ ], config::slmm slmm_parameters[ ], config::lasm lasm_parameters[ ], config::lalm lalm_parameters[ ], config::mt mt_parameters[ ], config::ChaoticMap map, int number_of_rounds)

  Initializes all chaotic map parameters to zero.

- static void pattern::assignMapParameters (config::lm lm_parameters[ ], config::lma lma_parameters[ ], config::slmm slmm_parameters[ ], config::lasm lasm_parameters[ ], config::lalm lalm_parameters[ ], config::mt mt_parameters[ ], config::ChaoticMap map, int number_of_rounds)

  Initializes all chaotic map parameters to random values.

- static void pattern::displayMapParameters (config::lm lm_parameters[ ], config::lma lma_parameters[ ], config::slmm slmm_parameters[ ], config::lasm lasm_parameters[ ], config::lalm lalm_parameters[ ], config::mt mt_parameters[ ], config::ChaoticMap map, int number_of_rounds)

  Displays all chaotic map parameters.

- static long pattern::rwMapParameters (config::lm lm_parameters[ ], config::lma lma_parameters[ ], config::slmm slmm_parameters[ ], config::lasm lasm_parameters[ ], config::lalm lalm_parameters[ ], config::mt mt_parameters[ ], config::ChaoticMap map, FILE *outfile, const char *mode, long ptr_position, int number_of_rounds)

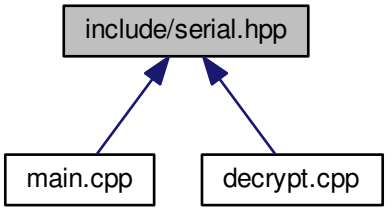
  Reads or writes parameters of chosen chaotic map.

- static void pattern::selectChaoticMap (config::lm lm_parameters[ ], config::lma lma_parameters[ ], config::slmm slmm_parameters[ ], config::lasm lasm_parameters[ ], config::lalm lalm_parameters[ ], config::mt mt_parameters[ ], double *x, double *y, double *x_bar, double *y_bar, uint32_t *&random_array, uint32_t *&lut_vec, config::ChaoticMap map, int iteration, uint32_t m, uint32_t random_array_length)

  Selects the chaotic map according to given chaotic map choice.

6.7 include/serial.hpp File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- serial

  This header file contains the Gray Level Transform function used for image diffusion.

Functions

- static void serial::grayLevelTransform (uint8_t ∗&img_vec, uint32_t ∗random_array, uint32_t total)

  Diffuses image vector with pseudorandom sequence.

6.8 kernel/kernel.cu File Reference

6.9 main.cpp File Reference

```
#include "include/commonheader.hpp"
#include "include/serial.hpp"
#include "include/pattern.hpp"
#include "include/kernel.hpp"
#include "include/io.hpp"
```

Include dependency graph for main.cpp:

Functions

- int main (int argc, char ∗argv[ ])

6.9.1 Function Documentation

6.9.1.1 int main (int *argc,* char * *argv[ ]*)

Get file path

Get image name from file path

Assign the number of rotation and swapping rounds

Parameter arrays

CPU vector declarations

GPU vector declarations

Allocating GPU memory

Flattening 2D N X M image to 1D N X M vector

Generating and swapping chaotic map lut

Assigning chaotic map choices for each vector

Initializing map parameters

Assigning map parameters

Writing number of rotation rounds

Writing number of swapping rounds

Writing seed used for shuffling the map_array

Writing map parameters

Display map parameters

Warming up GPU

Row and Column Rotation

Vector generation for row and column rotation

Row and Column Rotation

Copying CPU vectors to GPU memory

Getting results from GPU

Swapping img_vec and enc_vec

Row and Column Swapping

Vector generation for row and coulumn swapping

Transferring vectors from CPU to GPU memory

Getting results from GPU

Swapping enc_vec and final_vec

Diffusion

Generating diffusion array

Definition at line 8 of file main.cpp.

References pattern::assignMapParameters(), config::constant_parameters_file_path, DEBUG_INTERMEDIATE_IMAGES, DEBUG_MAP_CHOICES_ARRAY, DEBUG_MAP_PARAMETERS, DEBUG_READ_WRITE, DEBUG_VECTORS, DIFFUSION, pattern::displayMapParameters(), common::flattenImage(), common::genMapLUTVec(), common::getFileNameFromPath(), common::getRandomInteger(), serial::grayLevelTransform(), pattern::initializeMapParameters(), pattern::MTSequence(), common::printArray32(), common::printArray8(), ROW_COL_ROTATION, ROW_COL_SWAPPING, run_EncGenCatMap(), run_encRowColSwap(), run_WarmUp(), pattern::rwMapParameters(), pattern::selectChaoticMap(), and common::swapLUT().

```
9 {
10
14   std::string input_image_path = std::string(argv[1]);
15   cout<<"\nInput image = "<<input_image_path;
16
17
18   if(input_image_path == "")
19   {
20     cout<<"\nNo image name specified.Please specify an image name.\nExiting...";
21     exit(0);
22   }
23
27   std::string image_name = common::getFileNameFromPath(input_image_path);
28
29   auto start = std::chrono::system_clock::now();
30
31   cv::Mat image;
32   image = cv::imread(input_image_path,cv::IMREAD_UNCHANGED);
33
34   if(!image.data)
35   {
36     cout<<"\nCould not open image from "<<input_image_path<<" \nExiting...";
37     exit(0);
38   }
39
40   uint32_t m = 0,n = 0,channels = 0, total = 0;
41   int number_of_rotation_rounds = 0;
42   int number_of_swapping_rounds = 0;
43   int seed = 0;
44   int fseek_status = 9,fwrite_status = 9,i = 0;
45
46
47
48
49   long ptr_position = 0;
50   double alpha = 0.00,beta = 0.00;
51
52   m = image.rows;
53   n = image.cols;
54   channels = image.channels();
55   total = m * n;
56
57   cout<<"\nRows = "<<m;
58   cout<<"\nCols = "<<n;
59   cout<<"\nChannels = "<<channels;
60
61
65   number_of_rotation_rounds = common::getRandomInteger(5,5);
66   number_of_swapping_rounds = common::getRandomInteger(5,5);
67
68
69   config::ChaoticMap map_row_random_vec;
70   config::ChaoticMap map_col_random_vec;
71   config::ChaoticMap map_row_rotation_vec;
72   config::ChaoticMap map_col_rotation_vec;
73   config::ChaoticMap map_diffusion_array;
74
78   config::lm lm_parameters[6];
79   config::lma lma_parameters[6];
80   config::slmm slmm_parameters[6];
```

```
81   config::lasm lasm_parameters[6];
82   config::lalm lalm_parameters[6];
83   config::mt mt_parameters[6];
84
85
89   uint8_t *img_vec = (uint8_t*)calloc(total * channels,sizeof(uint8_t));
90   uint8_t *enc_vec = (uint8_t*)calloc(total * channels,sizeof(uint8_t));
91   uint8_t *final_vec = (uint8_t*)calloc(total * channels,sizeof(uint8_t));
92   uint32_t *row_swap_lut_vec = (uint32_t*)calloc(m,sizeof(uint32_t));
93   uint32_t *col_swap_lut_vec = (uint32_t*)calloc(n,sizeof(uint32_t));
94   uint32_t *U = (uint32_t*)calloc(m,sizeof(uint32_t));
95   uint32_t *V = (uint32_t*)calloc(n,sizeof(uint32_t));
96   double *x = (double*)calloc(total * channels,sizeof(double));
97   double *y = (double*)calloc(total * channels,sizeof(double));
98   double *x_bar = (double*)calloc(total * channels,sizeof(double));
99   double *y_bar = (double*)calloc(total * channels,sizeof(double));
100   uint32_t *row_rotation_vec = (uint32_t*)calloc(total * channels,sizeof(uint32_t));
101   uint32_t *col_rotation_vec = (uint32_t*)calloc(total * channels,sizeof(uint32_t));
102   uint32_t *row_random_vec = (uint32_t*)calloc(total * channels,sizeof(uint32_t));
103   uint32_t *col_random_vec = (uint32_t*)calloc(total * channels,sizeof(uint32_t));
104   uint32_t *diffusion_array = (uint32_t*)calloc(total * channels,sizeof(uint32_t));
105
106   uint32_t *dummy_lut_vec = (uint32_t*)calloc(2,sizeof(uint32_t));
107   uint32_t *map_array = (uint32_t*)calloc(15,sizeof(uint32_t));
108   uint32_t *map_choice_array = (uint32_t*)calloc(6,sizeof(uint32_t));
109
110
114   uint8_t *gpu_img_vec;
115   uint8_t *gpu_enc_vec;
116   uint8_t *gpu_final_vec;
117   const uint32_t *gpu_U;
118   const uint32_t *gpu_V;
119   const uint32_t *gpu_row_swap_lut_vec;
120   const uint32_t *gpu_col_swap_lut_vec;
121   const uint32_t *gpu_diffusion_array;
122
126   cudaMalloc((void**)&gpu_img_vec,total * channels * sizeof(uint8_t));
127   cudaMalloc((void**)&gpu_enc_vec,total * channels * sizeof(uint8_t));
128   cudaMalloc((void**)&gpu_final_vec,total * channels * sizeof(uint8_t));
129   cudaMalloc((void**)&gpu_U,m * sizeof(uint32_t));
130   cudaMalloc((void**)&gpu_V,n * sizeof(uint32_t));
131   cudaMalloc((void**)&gpu_row_swap_lut_vec,m * sizeof(uint32_t));
132   cudaMalloc((void**)&gpu_col_swap_lut_vec,n * sizeof(uint32_t));
133
137   common::flattenImage(image,img_vec,channels);
138
139   if(DEBUG_VECTORS == 1)
140   {
141     cout<<"\nplain image = ";
142     common::printArray8(img_vec,total * channels);
143   }
144
148   seed = common::getRandomInteger(1000,2000);
149   pattern::MTSequence(map_array,15,1,15,seed);
150   common::genMapLUTVec(map_choice_array,6);
151   common::swapLUT(map_choice_array,map_array,6);
152
156   map_row_random_vec = config::ChaoticMap(map_choice_array[0]);
157   map_col_random_vec = config::ChaoticMap(map_choice_array[1]);
158   map_row_rotation_vec = config::ChaoticMap(map_choice_array[2]);
159   map_col_rotation_vec = config::ChaoticMap(map_choice_array[3]);
160   map_diffusion_array = config::ChaoticMap(map_choice_array[4]);
161
162   if(DEBUG_MAP_CHOICES_ARRAY == 1)
163   {
164     cout<<"\nMap choices = ";
165     for(int i = 0 ; i < 6; ++i)
166     {
167       printf(" %d",map_choice_array[i]);
168     }
169   }
170
171
175   pattern::initializeMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_row_rotation_vec,number_of_rotation_rounds);
176   pattern::initializeMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_col_rotation_vec,number_of_rotation_rounds);
177   pattern::initializeMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_row_random_vec,number_of_swapping_rounds);
178   pattern::initializeMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_col_random_vec,number_of_swapping_rounds);
179   pattern::initializeMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_diffusion_array,1);
180
184   pattern::assignMapParameters(lm_parameters,lma_parameters,slmm_parameters,
      lasm_parameters,lalm_parameters,mt_parameters,map_row_rotation_vec,number_of_rotation_rounds);
185   pattern::assignMapParameters(lm_parameters,lma_parameters,slmm_parameters,
```

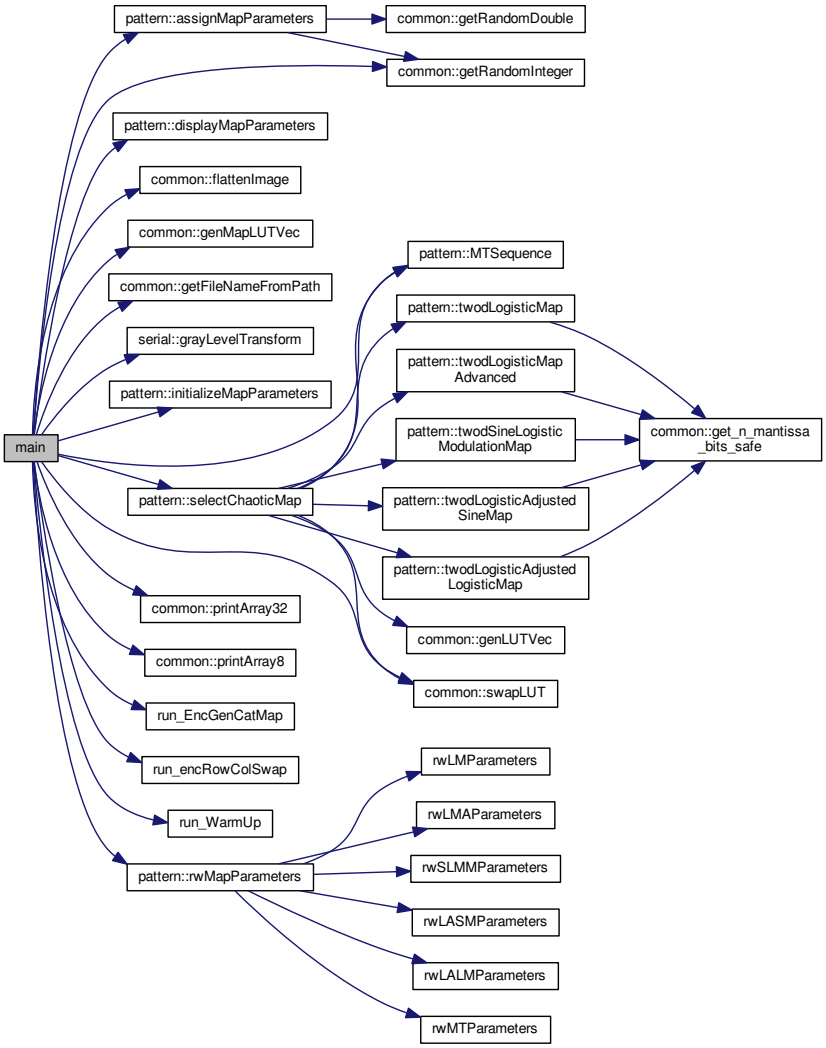
```
    lasm_parameters,lalm_parameters,mt_parameters,map_col_rotation_vec,number_of_rotation_rounds);
186 pattern::assignMapParameters(lm_parameters,lma_parameters,slmm_parameters,
    lasm_parameters,lalm_parameters,mt_parameters,map_row_random_vec,number_of_swapping_rounds);
187 pattern::assignMapParameters(lm_parameters,lma_parameters,slmm_parameters,
    lasm_parameters,lalm_parameters,mt_parameters,map_col_random_vec,number_of_swapping_rounds);
188 pattern::assignMapParameters(lm_parameters,lma_parameters,slmm_parameters,
    lasm_parameters,lalm_parameters,mt_parameters,map_diffusion_array,1);
189
193 FILE *outfile = fopen(config::constant_parameters_file_path,"wb");
194
195 if(outfile == NULL)
196 {
197   cout<<"\nCould not open "<<config::constant_parameters_file_path<<
    " for writing\nExiting...";
198   exit(0);
199 }
200
201 if(DEBUG_READ_WRITE == 1)
202 {
203   cout<<"\npointer position before writing the number of rotation rounds = "<<ptr_position;
204 }
205
206 fwrite_status = fwrite(&number_of_rotation_rounds,sizeof(number_of_rotation_rounds),1,outfile);
207 ptr_position =  ftell(outfile);
208
209
210 if(DEBUG_READ_WRITE == 1)
211 {
212   cout<<"\nfwrite status after writing the number of rotation rounds = "<<fwrite_status;
213   cout<<"\npointer position after writing the number of rotation rounds = "<<ptr_position;
214 }
215
216
217 fclose(outfile);
218
222 outfile = fopen(config::constant_parameters_file_path,"ab");
223
224 if(outfile == NULL)
225 {
226   cout<<"\nCould not open "<<config::constant_parameters_file_path<<
    " for writing\nExiting...";
227   exit(0);
228 }
229
230 if(ptr_position > 0)
231 {
232    fseek_status = fseek(outfile,(ptr_position + 1),SEEK_SET);
233    ptr_position = ftell(outfile);
234 }
235
236 if(DEBUG_READ_WRITE == 1)
237 {
238   cout<<"\npointer position before writing the number of swapping rounds = "<<ptr_position;
239 }
240
241 fwrite_status = fwrite(&number_of_swapping_rounds,sizeof(number_of_swapping_rounds),1,outfile);
242 ptr_position =  ftell(outfile);
243
244 if(DEBUG_READ_WRITE == 1)
245 {
246   cout<<"\nfwrite status after writing the number of swapping rounds = "<<fwrite_status;
247   cout<<"\npointer position after writing the number of swapping rounds = "<<ptr_position;
248 }
249
250 fclose(outfile);
251
252
256 outfile = fopen(config::constant_parameters_file_path,"ab");
257
258 if(outfile == NULL)
259 {
260   cout<<"\nCould not open "<<config::constant_parameters_file_path<<
    " for writing\nExiting...";
261   exit(0);
262 }
263
264 if(ptr_position > 0)
265 {
266    fseek_status = fseek(outfile,(ptr_position + 1),SEEK_SET);
267    ptr_position = ftell(outfile);
268 }
269
270 if(DEBUG_READ_WRITE == 1)
271 {
272   cout<<"\npointer position before writing seed = "<<ptr_position;
273 }
274
```

```
275   fwrite_status = fwrite(&seed,sizeof(seed),1,outfile);
276   ptr_position =  ftell(outfile);
277
278   if(DEBUG_READ_WRITE == 1)
279   {
280     cout<<"\nfwrite status after writing seed = "<<fwrite_status;
281     cout<<"\npointer position after writing seed = "<<ptr_position;
282   }
283
284   fclose(outfile);
285
286   if(DEBUG_MAP_PARAMETERS == 1)
287   {
288     cout<<"\nnumber of rotation rounds = "<<number_of_rotation_rounds;
289     cout<<"\nnumber of swapping rounds = "<<number_of_swapping_rounds;
290     cout<<"\nseed = "<<seed;
291   }
292
293
297   ptr_position = pattern::rwMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_row_rotation_vec,outfile,"ab",ptr_position,
      number_of_rotation_rounds);
298   ptr_position = pattern::rwMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_col_rotation_vec,outfile,"ab",ptr_position,
      number_of_rotation_rounds);
299   ptr_position = pattern::rwMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_row_random_vec,outfile,"ab",ptr_position,
      number_of_swapping_rounds);
300   ptr_position = pattern::rwMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_col_random_vec,outfile,"ab",ptr_position,
      number_of_swapping_rounds);
301   ptr_position = pattern::rwMapParameters(lm_parameters,lma_parameters,
      slmm_parameters,lasm_parameters,lalm_parameters,mt_parameters,map_diffusion_array,outfile,"ab",ptr_position,1);
302
303
304   if(DEBUG_MAP_PARAMETERS == 1)
305   {
309     pattern::displayMapParameters(lm_parameters,lma_parameters,slmm_parameters
      ,lasm_parameters,lalm_parameters,mt_parameters,map_row_rotation_vec,number_of_rotation_rounds);
310     pattern::displayMapParameters(lm_parameters,lma_parameters,slmm_parameters
      ,lasm_parameters,lalm_parameters,mt_parameters,map_col_rotation_vec,number_of_rotation_rounds);
311     pattern::displayMapParameters(lm_parameters,lma_parameters,slmm_parameters
      ,lasm_parameters,lalm_parameters,mt_parameters,map_row_random_vec,number_of_swapping_rounds);
312     pattern::displayMapParameters(lm_parameters,lma_parameters,slmm_parameters
      ,lasm_parameters,lalm_parameters,mt_parameters,map_col_random_vec,number_of_swapping_rounds);
313     pattern::displayMapParameters(lm_parameters,lma_parameters,slmm_parameters
      ,lasm_parameters,lalm_parameters,mt_parameters,map_diffusion_array,1);
314   }
315
319   dim3 warm_up_grid(1,1,1);
320   dim3 warm_up_block(1,1,1);
321   run_WarmUp(warm_up_grid,warm_up_block);
322
323   if(ROW_COL_ROTATION == 1)
324   {
329     for(i = 0; i < number_of_rotation_rounds; ++i)
330     {
335       pattern::selectChaoticMap(lm_parameters,lma_parameters,slmm_parameters,
      lasm_parameters,lalm_parameters,mt_parameters,x,y,x_bar,y_bar,row_rotation_vec,U,map_row_rotation_vec,i,m,total *
       channels);
336       pattern::selectChaoticMap(lm_parameters,lma_parameters,slmm_parameters,
      lasm_parameters,lalm_parameters,mt_parameters,x,y,x_bar,y_bar,col_rotation_vec,V,map_col_rotation_vec,i,n,total *
       channels);
337
341       dim3 enc_gen_cat_map_grid(m,n,1);
342       dim3 enc_gen_cat_map_blocks(channels,1,1);
343
347       cudaMemcpy(gpu_img_vec,img_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
348       cudaMemcpy(gpu_enc_vec,enc_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
349       cudaMemcpy((void*)gpu_U,U,m * sizeof(uint32_t),cudaMemcpyHostToDevice);
350       cudaMemcpy((void*)gpu_V,V,n * sizeof(uint32_t),cudaMemcpyHostToDevice);
351
352       run_EncGenCatMap(gpu_img_vec,gpu_enc_vec,(const uint32_t*)gpu_V,(const uint32_t*)
      gpu_U,enc_gen_cat_map_grid,enc_gen_cat_map_blocks);
353
357       cudaMemcpy(enc_vec,gpu_enc_vec,total * channels * sizeof(uint8_t),cudaMemcpyDeviceToHost);
358       cudaMemcpy(img_vec,gpu_img_vec,total * channels * sizeof(uint8_t),cudaMemcpyDeviceToHost);
359
363       if(number_of_rotation_rounds > 1)
364       {
365         cudaMemcpy(img_vec,enc_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToHost);
366         cudaMemcpy(gpu_img_vec,img_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
367       }
368
369
370       if(DEBUG_INTERMEDIATE_IMAGES == 1)
371       {
```

```
372         std::string rotated_image = "";
373         rotated_image = image_name + "_rotated" + "_ROUND_" + std::to_string(i + 1) + ".png";
374         cv::Mat img_reshape(m,n,image.type(),enc_vec);
375         bool rotate_status = cv::imwrite(rotated_image,img_reshape);
376 
377         if(rotate_status == 1)
378         {
379           cout<<"\nROTATION SUCCESSFUL FOR ROUND "<<i + 1;
380         }
381 
382         else
383         {
384           cout<<"\nROTATION UNSUCCESSFUL FOR ROUND "<<i + 1;
385         }
386       }
387 
388       if(DEBUG_VECTORS == 1)
389       {
390 
391         cout<<"\nimg_vec = ";
392         common::printArray8(img_vec,total * channels);
393 
394         printf("\nenc_vec = ");
395         common::printArray8(enc_vec,total * channels);
396 
397         cout<<"\nU = ";
398         common::printArray32(U,m);
399 
400         cout<<"\nV = ";
401         common::printArray32(V,n);
402 
403         cout<<"\nRow rotation vec = ";
404         common::printArray32(row_rotation_vec,total * channels);
405 
406         cout<<"\nColumn rotation vec = ";
407         common::printArray32(col_rotation_vec,total * channels);
408       }
409 
410 
411 
412     }
413   }
414 
415   if(ROW_COL_SWAPPING == 1)
416   {
421     for(i = 0; i < number_of_swapping_rounds; ++i)
422     {
427       pattern::selectChaoticMap(lm_parameters,lma_parameters,slmm_parameters,
      lasm_parameters,lalm_parameters,mt_parameters,x,y,x_bar,y_bar,row_random_vec,row_swap_lut_vec,map_row_random_vec,
      i,m,total * channels);
428       pattern::selectChaoticMap(lm_parameters,lma_parameters,slmm_parameters,
      lasm_parameters,lalm_parameters,mt_parameters,x,y,x_bar,y_bar,col_random_vec,col_swap_lut_vec,map_col_random_vec,
      i,n,total * channels);
429 
430       dim3 enc_row_col_swap_grid(m,n,1);
431       dim3 enc_row_col_swap_blocks(channels,1,1);
432 
436       cudaMemcpy(gpu_enc_vec,enc_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
437       cudaMemcpy(gpu_final_vec,final_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
438       cudaMemcpy((void*)gpu_row_swap_lut_vec,row_swap_lut_vec,m * sizeof(uint32_t),cudaMemcpyHostToDevice);
439       cudaMemcpy((void*)gpu_col_swap_lut_vec,col_swap_lut_vec,n * sizeof(uint32_t),cudaMemcpyHostToDevice);
440 
441       run_encRowColSwap(gpu_enc_vec,gpu_final_vec,(const uint32_t *)gpu_row_swap_lut_vec,(
      const uint32_t*)gpu_col_swap_lut_vec,enc_row_col_swap_grid,enc_row_col_swap_blocks);
442 
446       cudaMemcpy(final_vec,gpu_final_vec,total * channels * sizeof(uint8_t),cudaMemcpyDeviceToHost);
447       cudaMemcpy(enc_vec,gpu_enc_vec,total * channels * sizeof(uint8_t),cudaMemcpyDeviceToHost);
448 
452       if(number_of_swapping_rounds > 1)
453       {
454         cudaMemcpy(enc_vec,final_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToHost);
455         cudaMemcpy(gpu_enc_vec,enc_vec,total * channels * sizeof(uint8_t),cudaMemcpyHostToDevice);
456       }
457 
458 
459       if(DEBUG_VECTORS == 1)
460       {
461         cout<<"\n\ni = "<<i;
462         printf("\nenc_vec = ");
463 
464         common::printArray8(enc_vec,total * channels);
465         printf("\nfinal_vec = ");
466 
467         common::printArray8(final_vec,total * channels);
468         cout<<"\n\ni = "<<i;
469 
```

```
470         printf("\nrow_swap_lut_vec = ");
471         common::printArray32(row_swap_lut_vec,m);
472 
473         printf("\ncol_swap_lut_vec = ");
474         common::printArray32(col_swap_lut_vec,n);
475 
476       }
477 
478       if(DEBUG_INTERMEDIATE_IMAGES == 1)
479       {
480 
481         std::string swapped_image = "";
482         swapped_image = image_name + "_swapped" + "_ROUND_" + std::to_string(i + 1) + ".png";
483         cv::Mat img_reshape(m,n,image.type(),final_vec);
484         bool swap_status = cv::imwrite(swapped_image,img_reshape);
485 
486         if(swap_status == 1)
487         {
488           cout<<"\nSWAPPING SUCCESSFUL FOR "<<"ROUND "<<i + 1;
489         }
490 
491         else
492         {
493           cout<<"\nSWAPPING UNSUCCESSFUL FOR "<<"ROUND "<<i + 1;
494         }
495 
496       }
497 
498     }
499   }
500 
505   if(DIFFUSION == 1)
506   {
507 
511       pattern::selectChaoticMap(lm_parameters,lma_parameters,slmm_parameters,
    lasm_parameters,lalm_parameters,mt_parameters,x,y,x_bar,y_bar,diffusion_array,dummy_lut_vec,map_diffusion_array,0
    ,2,total * channels);
512 
513       auto diffusion_start = std::chrono::system_clock::now();
514 
515       serial::grayLevelTransform(final_vec,diffusion_array,total * channels);
516 
517       auto diffusion_end = std::chrono::system_clock::now();
518 
519       auto diffusion_time = std::chrono::duration_cast<std::chrono::milliseconds>(diffusion_end -
    diffusion_start).count();
520       cout<<"\nDiffusion time ="<<diffusion_time <<" ms";
521 
522     if(DEBUG_VECTORS == 1)
523     {
524       printf("\n\nRotated Swapped and Diffused image = ");
525       common::printArray8(final_vec,total * channels);
526     }
527 
528     if(DEBUG_INTERMEDIATE_IMAGES == 1)
529     {
530       std::string diffused_image  = "";
531       diffused_image = image_name + "_diffused" + ".png";
532       cv::Mat img_reshape(m,n,image.type(),final_vec);
533       bool diffusion_status = cv::imwrite(diffused_image,img_reshape);
534 
535       if(diffusion_status == 1)
536       {
537         cout<<"\nDIFFUSION SUCCESSFUL";
538 
539       }
540 
541       else
542       {
543         cout<<"\nDIFFUSION UNSUCCESSFUL";
544       }
545 
546     }
547 
548   }
549    auto end = std::chrono::system_clock::now();
550 
551    auto elapsed = std::chrono::duration_cast<std::chrono::microseconds>(end - start).count();
552 
553    cout<<"\nTotal encryption time  = "<<elapsed / 1000000<<" s";
554 
555    return 0;
556 }
```

Here is the call graph for this function:



Index