# ARGON2

Generated by Doxygen 1.8.13

# Contents

# Chapter 1

# Argon2

Argon2_AdvancedProgramming16/17

Per compilare:

$ make

make options:

(°) debug = 1 -> compile with flag -ggdb, creating the table of symbols (°) follow-specifications = 1 -> compile with -DFOLLOW_SPECIFICATION, strictly following the Argon2 specification

exa.: $ make debug=1

Regole del makefile:

Per creare un main di test, che compari l'hash generato con la versione ufficiale della phc release:

$ make test

Per creare una versione di argon2 o del test per argon2 su cui fare un'analisi della memoria usata:

$ make bad_memory

o

$ make test_bad_memory

Per creare un benchmark che valuti le prestazioni di Argon2 su diversi parametri:

$ make bench

Per cancellare i file oggetto e altri file evenutalmente creati:

$ make clean

Per eliminare completamente Argon2:

$ make purge

# Chapter 2

# Data Structure Index

## 2.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

# Chapter 4

# Data Structure Documentation

## 4.1 Argon2_arguments Struct Reference

`#include <Argon2_body.h>`

### Data Fields

- uint8_t * P
- uint32_t size_P
- uint8_t * S

  *Nonce S.*
- uint32_t size_S

  *nonce size, in [8 .. 2^32-1]*
- uint32_t p

  *Degree of parallelization, in [1 .. 2^24-1].*
- uint32_t tau

  *Tag length.*
- uint32_t m

  *Total number of memory blocks.*
- uint32_t t

  *Number of steps.*
- uint32_t v

  *Version byte, default = 0x13.*
- uint8_t * K

  *Key K.*
- uint32_t size_K

  *Key size, in [0 .. 2^32].*
- uint8_t * X

  *Associated data X.*
- uint32_t size_X

  *Associated data size, in [0..2^32].*
- uint32_t y

  *Type value, defining the version of Argon2 : 0 = d , 1 = i , 2 = id , 4 = ds.*

### 4.1.1 Field Documentation

#### 4.1.1.1 K

```
uint8_t* Argon2_arguments::K
```

Key K.

#### 4.1.1.2 m

```
uint32_t Argon2_arguments::m
```

Total number of memory blocks.

#### 4.1.1.3 P

```
uint8_t* Argon2_arguments::P
```

#### 4.1.1.4 p

```
uint32_t Argon2_arguments::p
```

Degree of parallelization, in [1 .. 2^24-1].

#### 4.1.1.5 S

```
uint8_t* Argon2_arguments::S
```

Nonce S.

#### 4.1.1.6 size_K

```
uint32_t Argon2_arguments::size_K
```

Key size, in [0 .. 2^32].

#### 4.1.1.7 size_P

```
uint32_t Argon2_arguments::size_P
```

#### 4.1.1.8 size_S

```
uint32_t Argon2_arguments::size_S
```

nonce size, in [8 .. 2^32-1]

#### 4.1.1.9 size_X

```
uint32_t Argon2_arguments::size_X
```

Associated data size, in [0..2^32].

#### 4.1.1.10 t

```
uint32_t Argon2_arguments::t
```

Number of steps.

#### 4.1.1.11 tau

```
uint32_t Argon2_arguments::tau
```

Tag length.

#### 4.1.1.12 v

```
uint32_t Argon2_arguments::v
```

Version byte, default = 0x13.

#### 4.1.1.13 X

```
uint8_t* Argon2_arguments::X
```

Associated data X.

#### 4.1.1.14 y

```
uint32_t Argon2_arguments::y
```

Type value, defining the version of Argon2 : 0 = d , 1 = i , 2 = id , 4 = ds.

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

- Argon2_body.h

## 4.2 Argon2_global_workspace Struct Reference

```
#include <Argon2_matrix.h>
```

### Data Fields

- uint64_t * matrix
- uint32_t p
  *degree of parallelism*
- uint32_t q
  *number of columns in the matrix*
- uint64_t r
  *pass number*
- uint64_t m
  *Total memory blocks.*
- uint64_t s
  *Slice number.*
- uint64_t t
  *total passes number*
- uint64_t x
  *type number*
- uint32_t segment_length
- uint64_t * S
  *S-Box for Argon2ds.*

### 4.2.1 Field Documentation

#### 4.2.1.1 m

```
uint64_t Argon2_global_workspace::m
```

Total memory blocks.

#### 4.2.1.2 matrix

```
uint64_t* Argon2_global_workspace::matrix
```

#### 4.2.1.3 p

```
uint32_t Argon2_global_workspace::p
```

degree of parallelism

#### 4.2.1.4 q

```
uint32_t Argon2_global_workspace::q
```

number of columns in the matrix

#### 4.2.1.5 r

```
uint64_t Argon2_global_workspace::r
```

pass number

#### 4.2.1.6 s

```
uint64_t Argon2_global_workspace::s
```

Slice number.

#### 4.2.1.7 S

```
uint64_t* Argon2_global_workspace::S
```

S-Box for Argon2ds.

#### 4.2.1.8 segment_length

```
uint32_t Argon2_global_workspace::segment_length
```

#### 4.2.1.9 t

```
uint64_t Argon2_global_workspace::t
```

total passes number

#### 4.2.1.10 x

```
uint64_t Argon2_global_workspace::x
```

type number

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

- Argon2_matrix.h

## 4.3 Argon2_local_workspace Struct Reference

```
#include <Argon2_matrix.h>
```

### Data Fields

- uint64_t l
  *lane number*
- uint64_t c
  *column number*
- uint64_t i
  *counter [reset for every segment]*
- uint64_t pairs [A2I_PAIRS_NUMBER]
  *place to save the 128 pairs in argon2i*
- uint64_t counter
  *used pairs counter*

### 4.3.1 Field Documentation

#### 4.3.1.1 c

```
uint64_t Argon2_local_workspace::c
```

column number

#### 4.3.1.2 counter

```
uint64_t Argon2_local_workspace::counter
```

used pairs counter

#### 4.3.1.3 i

```
uint64_t Argon2_local_workspace::i
```

counter [reset for every segment]

#### 4.3.1.4 l

```
uint64_t Argon2_local_workspace::l
```

lane number

#### 4.3.1.5 pairs

```
uint64_t Argon2_local_workspace::pairs[A2I_PAIRS_NUMBER]
```

place to save the 128 pairs in argon2i

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

- Argon2_matrix.h

# Chapter 5

# File Documentation

## 5.1 Argon2.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include "Argon2_body.h"
```

Include dependency graph for Argon2.c:

## 5.2 Argon2_body.c File Reference

```
#include "Argon2_body.h"
```

Include dependency graph for Argon2_body.c:

### Macros

- #define CAT_N(array, pointer, n) {memcpy(array,pointer,n); array+=n;}

### Functions

- void compute_H0 (Argon2_arguments *args, uint8_t *H0)
- void compute_first_block (Argon2_global_workspace *B, uint8_t *H0, uint32_t tau, uint32_t c)
- void compute_segment (Argon2_global_workspace *B, Argon2_local_workspace *args)
- void perform_step (Argon2_global_workspace *B)
- void finalize (Argon2_global_workspace *B, uint64_t *B_final)
- void Argon2 (Argon2_arguments *args, uint8_t *tag)

#### 5.2.1 Detailed Description

core functions of Argon2

### 5.2.2 Macro Definition Documentation

#### 5.2.2.1 CAT_N

```
#define CAT_N(
              array,
              pointer,
              n ) {memcpy(array,pointer,n); array+=n;}
```

Concatenation of N bytes to the array. It also handles the update of the pointer to the tail of the array

### 5.2.3 Function Documentation

#### 5.2.3.1 Argon2()

```
void Argon2 (
             Argon2_arguments * args,
             uint8_t * tag )
```

Initializes the global environment, performs computations and stores the output in tag

**Parameters**

| | |
|---|---|
| *args* | pointer to the arguments for Argon2 to be inizialized |
| *tag* | pointer to the tag |

#### 5.2.3.2 compute_first_block()

```
void compute_first_block (
             Argon2_global_workspace * B,
             uint8_t * H0,
             uint32_t tau,
             uint32_t c )
```

Initialization of the first two columns [c = 0,1] of the matrix, using the seed H0

**Parameters**

| | |
|---|---|
| *B* | pointer to the memory matrix used for data storage in Argon2 |
| *H0* | pointer to H0, initial seed for the first block computation |
| *tau* | tag length |
| *c* | column index |

#### 5.2.3.3 compute_H0()

```
void compute_H0 (
            Argon2_arguments * args,
            uint8_t * H0 )
```

Computes the seed for the initialization of the first two columns in the first step of Argon2

**Parameters**

| | |
|---|---|
| *args* | pointer to the arguments for Argon2 to be inizialized |
| *H0* | pointer to H0, initial seed for the first block computation |

#### 5.2.3.4 compute_segment()

```
void compute_segment (
            Argon2_global_workspace * B,
            Argon2_local_workspace * args )
```

Computes all the blocks in a segment

**Parameters**

| | |
|---|---|
| *B* | pointer to the memory matrix used for data storage in Argon2 |
| *args* | pointer to the arguments for Argon2 to be inizialized |

#### 5.2.3.5 finalize()

```
void finalize (
            Argon2_global_workspace * B,
            uint64_t * B_final )
```

Function to get the final block. Remark: B_final needs to be whitened

**Parameters**

| | |
|---|---|
| *B* | pointer to the memory matrix used for data storage in Argon2 |
| *B_final* | pointer to the pre-hashing digest of Argon2 |

#### 5.2.3.6 perform_step()

```
void perform_step (
            Argon2_global_workspace * B )
```

Initializes arguments and handles parallel computation in an Argon2 step.

**Parameters**

| | |
|---|---|
| *B* | pointer to the memory matrix used for data storage in Argon2 |

## 5.3 Argon2_body.h File Reference

`#include "Argon2_matrix.h"`
Include dependency graph for Argon2_body.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct Argon2_arguments

### Macros

- #define H0_LENGTH 64

### Functions

- void Argon2 (Argon2_arguments *args, uint8_t *tag)

### 5.3.1 Detailed Description

Interface for Argon2

### 5.3.2 Macro Definition Documentation

#### 5.3.2.1 H0_LENGTH

```
#define H0_LENGTH 64
```

length of the initial seed for first block computation

### 5.3.3 Function Documentation

#### 5.3.3.1 Argon2()

```
void Argon2 (
            Argon2_arguments * args,
            uint8_t * tag )
```

Initializes the global environment, performs computations and stores the output in tag

**Parameters**

| | |
|---|---|
| *args* | pointer to the arguments for Argon2 to be inizialized |
| *tag* | pointer to the tag |

## 5.4 Argon2_compression.c File Reference

`#include "Argon2_compression.h"`
Include dependency graph for Argon2_compression.c:

### Macros

- #define A2DS_F {for (int k = 0; k < 8; ++k) P(block_00+16∗k);}

### Functions

- void XOR_128 (const uint64_t ∗X, const uint64_t ∗Y, uint64_t ∗res)
- void Core_G (uint64_t ∗a, uint64_t ∗b, uint64_t ∗c, uint64_t ∗d)
- void P (uint64_t ∗S)
- void S_Box_Inizialization (uint64_t ∗block_00, uint64_t ∗S)
- uint64_t Tau (uint64_t W, uint64_t ∗S)
- void A2DS_compression (uint64_t ∗R, uint64_t ∗Z, uint64_t ∗S)
- void A2_G (const uint64_t ∗X, const uint64_t ∗Y, uint64_t ∗result, uint64_t ∗S, uint8_t type)
- void H_prime (uint8_t ∗X, uint32_t sizeX, uint32_t tau, uint8_t ∗digest)

### 5.4.1 Detailed Description

Compression function of Argon2. It is built upon the function P taken from Blake2b

### 5.4.2 Macro Definition Documentation

#### 5.4.2.1 A2DS_F

```
#define A2DS_F {for (int k = 0; k < 8; ++k) P(block_00+16*k);}
```

Round function for the S-Box initialization Argon2ds

### 5.4.3 Function Documentation

#### 5.4.3.1 A2_G()

```
void A2_G (
            const uint64_t * X,
            const uint64_t * Y,
            uint64_t * result,
            uint64_t * S,
            uint8_t type )
```

Compression functions of Argon2 G : (X,Y) -> R = X ^ Y -> Q -> Z -> Z ^ R

**Parameters**

| | |
|---|---|
| *X* | pointer to the first input of the compression function |
| *Y* | pointer to the second input of the compression function |
| *result* | pointer to the result of the compression function |
| *S* | pointer to the image of the S box |
| *type* | version of Argon2 to be used |

#### 5.4.3.2 A2DS_compression()

```
void A2DS_compression (
            uint64_t * R,
            uint64_t * Z,
            uint64_t * S )
```

Extra computation required in the compression function for the 2ds version

**Parameters**

| | |
|---|---|
| *R* | pointer to R defined in the compression function |
| *Z* | pointer to Z defined in the compression function |
| *S* | pointer to the image of the S box |

#### 5.4.3.3 Core_G()

```
void Core_G (
            uint64_t * a,
            uint64_t * b,
            uint64_t * c,
            uint64_t * d )
```

Slightly modified version of the function B2B_G in Blake2b It is the core function for the permutation P

**Parameters**

| | |
|---|---|
| *a* | pointer to one of the input of the core of the round function of Blake2b |
| *b* | pointer to one of the input of the core of the round function of Blake2b |
| *c* | pointer to one of the input of the core of the round function of Blake2b |
| *d* | pointer to one of the input of the core of the round function of Blake2b |

#### 5.4.3.4 H_prime()

```
void H_prime (
            uint8_t * X,
            uint32_t sizeX,
            uint32_t tau,
            uint8_t * digest )
```

Variable-lenght hash function based on Blake2b tau is the lenght of the digest

**Parameters**

| | |
|---|---|
| *X* | pointer to the input of Argon2 hash function |
| *sizex* | size of the input |
| *tau* | length of the digest |
| *digest* | pointer to the resulting digest |

#### 5.4.3.5 P()

```
void P (
            uint64_t * S )
```

Slightly modified version of round function of Blake2b it takes as input the address of an array cointaining 16 uint64_t

**Parameters**

| | |
|---|---|
| *S* | pointer to input of the round function of Blake2b |

#### 5.4.3.6 S_Box_Inizialization()

```
void S_Box_Inizialization (
            uint64_t * block_00,
            uint64_t * S )
```

Takes the block at position (0,0) and initializes the S-Box S for Argon2ds

**Parameters**

| | |
|---|---|
| *block_00* | pointer to the block in position [0,0] of the matrix B |
| *S* | pointer to the image of the S box |

#### 5.4.3.7 Tau()

```
uint64_t Tau (
            uint64_t W,
            uint64_t * S )
```

64-bit transformation involved in the 2ds version

**Parameters**

| | |
|---|---|
| *W* | 64-bit word |
| *S* | pointer to the image of the S box |

#### 5.4.3.8 XOR_128()

```
void XOR_128 (
            const uint64_t * X,
            const uint64_t * Y,
            uint64_t * res )
```

Utility function performing the componentwise xor of two arrays

**Parameters**

| | |
|---|---|
| *X* | pointer to the first input array |
| *Y* | pointer to the second input array |
| *res* | pointer to the result of the xor |

## 5.5 Argon2_compression.h File Reference

`#include "Blake2b.h"`
Include dependency graph for Argon2_compression.h: This graph shows which files directly or indirectly include this file:

### Macros

- #define A2D 0
- #define A2I 1
- #define A2ID 2
- #define A2DS 4
- #define TRUNC_32(m) (m & 0x00000000FFFFFFFF)

### Functions

- void S_Box_Inizialization (uint64_t ∗block_00, uint64_t ∗S)
- void A2_G (const uint64_t ∗X, const uint64_t ∗Y, uint64_t ∗result, uint64_t ∗S, uint8_t type)
- void H_prime (uint8_t ∗X, uint32_t sizeX, uint32_t tau, uint8_t ∗digest)
- void XOR_128 (const uint64_t ∗X, const uint64_t ∗Y, uint64_t ∗res)

### 5.5.1 Detailed Description

Interface for the compression process in Argon2

### 5.5.2 Macro Definition Documentation

#### 5.5.2.1 A2D

```
#define A2D 0
```

type number for Argon2d

#### 5.5.2.2 A2DS

```
#define A2DS 4
```

type number for Argon2ds

#### 5.5.2.3 A2I

```
#define A2I 1
```

type number for Argon2i

#### 5.5.2.4 A2ID

```
#define A2ID 2
```

type number for Argon2id

#### 5.5.2.5 TRUNC_32

```
#define TRUNC_32(
              m ) (m & 0x00000000FFFFFFFF)
```

### 5.5.3 Function Documentation

#### 5.5.3.1 A2_G()

```
void A2_G (
             const uint64_t * X,
             const uint64_t * Y,
             uint64_t * result,
             uint64_t * S,
             uint8_t type )
```

Compression functions of Argon2 G : (X,Y) -> R = X ^ Y -> Q -> Z -> Z ^ R

**Parameters**

| | |
|---|---|
| *X* | pointer to the first input of the compression function |
| *Y* | pointer to the second input of the compression function |
| *result* | pointer to the result of the compression function |
| *S* | pointer to the image of the S box |
| *type* | version of Argon2 to be used |

#### 5.5.3.2 H_prime()

```
void H_prime (
             uint8_t * X,
             uint32_t sizeX,
             uint32_t tau,
             uint8_t * digest )
```

Variable-lenght hash function based on Blake2b tau is the lenght of the digest

**Parameters**

| | |
|---|---|
| *X* | pointer to the input of Argon2 hash function |
| *sizex* | size of the input |
| *tau* | length of the digest |
| *digest* | pointer to the resulting digest |

#### 5.5.3.3 S_Box_Inizialization()

```
void S_Box_Inizialization (
             uint64_t * block_00,
             uint64_t * S )
```

Takes the block at position (0,0) and initializes the S-Box S for Argon2ds

**Parameters**

| | |
|---|---|
| *block_00* | pointer to the block in position [0,0] of the matrix B |
| *S* | pointer to the image of the S box |

#### 5.5.3.4 XOR_128()

```
void XOR_128 (
             const uint64_t * X,
```

```
            const uint64_t * Y,
            uint64_t * res )
```

Utility function performing the componentwise xor of two arrays

**Parameters**

| | |
|---|---|
| *X* | pointer to the first input array |
| *Y* | pointer to the second input array |
| *res* | pointer to the result of the xor |

## 5.6 Argon2_matrix.c File Reference

`#include "Argon2_matrix.h"`
Include dependency graph for Argon2_matrix.c:

### Functions

- int Argon2_global_workspace_init (uint32_t m, uint32_t p, uint32_t t, uint32_t x, Argon2_global_workspace ∗B)
- int Argon2_matrix_get_block (uint32_t i, uint32_t j, uint64_t ∗∗dst, Argon2_global_workspace ∗src)
- void Argon2_global_workspace_free (Argon2_global_workspace ∗B)
- uint64_t Argon2d_generate_values (Argon2_global_workspace ∗B, uint32_t i, uint32_t j)
- void Argon2i_generate_values (Argon2_global_workspace ∗B, Argon2_local_workspace ∗args)
- uint64_t Argon2_indexing_mapping (Argon2_local_workspace ∗arg, Argon2_global_workspace ∗B, uint64↩ _t J)
- uint64_t Argon2_indexing (Argon2_global_workspace ∗B, Argon2_local_workspace ∗arg)

### 5.6.1 Detailed Description

Manages memory used in Argon2, with particular care to block indexing and safe inizialization and destruction

### 5.6.2 Function Documentation

#### 5.6.2.1 Argon2_global_workspace_free()

```
void Argon2_global_workspace_free (
            Argon2_global_workspace * B )
```

Deallocates the memory of the matrix and, if it is the case, also the memory used for the S-Box

Safely deallocates memory used in the global workspace

#### 5.6.2.2 Argon2_global_workspace_init()

```
int Argon2_global_workspace_init (
             uint32_t m,
             uint32_t p,
             uint32_t t,
             uint32_t x,
             Argon2_global_workspace * B )
```

Initializes the matrix and sets its parameters

**Parameters**

| | |
|---|---|
| *m* | KiB of memory to be used rounded down to the closest multiple of 4p |
| *p* | maximum degree of parallelism |
| *t* | total number of passes |
| *x* | type of Argon function |
| *B* | pointer to the memory matrix used for data storage in Argon2 |

Initializes the global workspace, allocating space for the matrix and setting parameters according to Argon2 input

#### 5.6.2.3 Argon2_indexing()

```
uint64_t Argon2_indexing (
            Argon2_global_workspace * B,
            Argon2_local_workspace * arg )
```

Indexing function, computes (i',j'), given the current position

#### 5.6.2.4 Argon2_indexing_mapping()

```
uint64_t Argon2_indexing_mapping (
            Argon2_local_workspace * arg,
            Argon2_global_workspace * B,
            uint64_t J )
```

Maps the values J1, J2 into i',j', indeces of a referenciable block

#### 5.6.2.5 Argon2_matrix_get_block()

```
int Argon2_matrix_get_block (
            uint32_t i,
            uint32_t j,
            uint64_t ** dst,
            Argon2_global_workspace * src )
```

Gets the block in position (i,j) in the Argon2 matrix in global_workspace, storing it in dst

Gets the block in position (i,j) in the Argon2 matrix B, storing a pointer to it in dst

#### 5.6.2.6 Argon2d_generate_values()

```
uint64_t Argon2d_generate_values (
            Argon2_global_workspace * B,
            uint32_t i,
            uint32_t j )
```

Generates J1, J2 as specified in Argon2d data dependent indexing

#### 5.6.2.7 Argon2i_generate_values()

```
void Argon2i_generate_values (
            Argon2_global_workspace * B,
            Argon2_local_workspace * args )
```

Generates 128 pairs (J1,J2) t.b.u. in the data independent indexing function for Argon2i and Argon2id

## 5.7 Argon2_matrix.h File Reference

`#include "Argon2_compression.h"`
Include dependency graph for Argon2_matrix.h: This graph shows which files directly or indirectly include this file:

### Data Structures

- struct Argon2_global_workspace
- struct Argon2_local_workspace

### Macros

- #define A2_MATRIX_BLOCK_LENGTH 1024
- #define A2I_PAIRS_NUMBER 128

### Functions

- int Argon2_global_workspace_init (uint32_t m, uint32_t p, uint32_t t, uint32_t x, Argon2_global_workspace ∗B)
- int Argon2_matrix_get_block (uint32_t i, uint32_t j, uint64_t ∗∗dst, Argon2_global_workspace ∗src)
- uint64_t Argon2_indexing (Argon2_global_workspace ∗B, Argon2_local_workspace ∗arg)
- void Argon2_global_workspace_free (Argon2_global_workspace ∗B)

### 5.7.1 Detailed Description

Interface for memory management and context separation for parallel computation

### 5.7.2 Macro Definition Documentation

#### 5.7.2.1 A2_MATRIX_BLOCK_LENGTH

```
#define A2_MATRIX_BLOCK_LENGTH 1024
```

#### 5.7.2.2 A2I_PAIRS_NUMBER

```
#define A2I_PAIRS_NUMBER 128
```

### 5.7.3 Function Documentation

#### 5.7.3.1 Argon2_global_workspace_free()

```
void Argon2_global_workspace_free (
            Argon2_global_workspace * B )
```

Deallocates the memory of the matrix and, if it is the case, also the memory used for the S-Box

Safely deallocates memory used in the global workspace

#### 5.7.3.2 Argon2_global_workspace_init()

```
int Argon2_global_workspace_init (
            uint32_t m,
            uint32_t p,
            uint32_t t,
            uint32_t x,
            Argon2_global_workspace * B )
```

Initializes the matrix and sets its parameters

**Parameters**

| | |
|---|---|
| *m* | KiB of memory to be used rounded down to the closest multiple of 4p |
| *p* | maximum degree of parallelism |
| *t* | total number of passes |
| *x* | type of Argon function |
| *B* | pointer to the memory matrix used for data storage in Argon2 |

Initializes the global workspace, allocating space for the matrix and setting parameters according to Argon2 input

#### 5.7.3.3 Argon2_indexing()

```
uint64_t Argon2_indexing (
            Argon2_global_workspace * B,
            Argon2_local_workspace * arg )
```

Indexing function, computes (i',j'), given the current position

#### 5.7.3.4 Argon2_matrix_get_block()

```
int Argon2_matrix_get_block (
            uint32_t i,
            uint32_t j,
            uint64_t ** dst,
            Argon2_global_workspace * src )
```

Gets the block in position (i,j) in the Argon2 matrix in global_workspace, storing it in dst

Gets the block in position (i,j) in the Argon2 matrix B, storing a pointer to it in dst

## 5.8 bench.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "Argon2_body.h"
```

Include dependency graph for bench.c:

### Functions

- int main ()

### Variables

- const char * types [ ] = {"Argon2d","Argon2i","Argon2id","Argon2s"}

### 5.8.1 Detailed Description

Benchmark for comparison with the official phc release of Argon2, it uses the same parameters used in the official phc implementation, in order to have a consistent test.

### 5.8.2 Function Documentation

#### 5.8.2.1 main()

```
int main ( )
```

Performs the benchmark, in particular we use the same parameters used in the official benchmark:
(°) Used memory: form 1 MiB up to 4 GiB;
(°) Degree of parallelization: from 1 to 8;
(°) ALl the four types of Argon2.

### 5.8.3 Variable Documentation

#### 5.8.3.1 types

```
types = {"Argon2d","Argon2i","Argon2id","Argon2s"}
```

Used to store the names of Argon2 types for a nice formatted output

## 5.9 Blake2b.c File Reference

```
#include "Blake2b.h"
```
Include dependency graph for Blake2b.c:

### Macros

- #define WORD_LENGTH 64
- #define ROUNDS_NUMER 12
- #define BLOCK_LENGTH 128
- #define WORKSPACE_LENGTH 8
- #define R1 32
- #define R2 24
- #define R3 16
- #define R4 63

### Functions

- void B2B_G (uint64_t *v, int a, int b, int c, int d, uint64_t x, uint64_t y)
- void B2B_F (uint64_t *h, uint64_t *m, uint64_t *t, int f)
- void blake2b (uint8_t *digest, size_t digest_size, uint8_t *data, uint64_t data_size)

### 5.9.1 Detailed Description

Ad hoc version of Blake2b, purged of all the operations that are not required for the Argon2 specification

### 5.9.2 Macro Definition Documentation

#### 5.9.2.1 BLOCK_LENGTH

```
#define BLOCK_LENGTH 128
```

Length of a data block to be compressed

#### 5.9.2.2 R1

```
#define R1 32
```

Constant offset for the first rotational shift

#### 5.9.2.3 R2

```
#define R2 24
```

Constant offset for the second rotational shift

#### 5.9.2.4 R3

```
#define R3 16
```

Constant offset for the third rotational shift

#### 5.9.2.5 R4

```
#define R4 63
```

Constant offset for the fourth rotational shift

#### 5.9.2.6 ROUNDS_NUMER

```
#define ROUNDS_NUMER 12
```

Number of rounds in the blake2b compression function

#### 5.9.2.7 WORD_LENGTH

```
#define WORD_LENGTH 64
```

Length of the word used for workspace initialization, as well as compressed data storage

#### 5.9.2.8 WORKSPACE_LENGTH

```
#define WORKSPACE_LENGTH 8
```

Dimension of the workspace used in the compression function, it is the size of a word, when expressed in octets of bytes

### 5.9.3 Function Documentation

#### 5.9.3.1 B2B_F()

```
void B2B_F (
             uint64_t * h,
             uint64_t * m,
             uint64_t * t,
             int f )
```

Main compression function of Blake2b, compresses the data contained in m into h

**Parameters**

| | |
|---|---|
| *h* | a word of length WORD_LENGTH for workspace initialization and compressed data storage |
| *m* | data block of length BLOCK_LENGTH, taken from the input data for blak2b |
| *t* | the counter of the data offset, it is a 128 bits unsigned integer, encoded in little endian as an array of two uint64_t |
| *f* | 1 if this is the final block to be encoded, 0 otherwise |

#### 5.9.3.2 B2B_G()

```
void B2B_G (
            uint64_t * v,
            int a,
            int b,
            int c,
            int d,
            uint64_t x,
            uint64_t y )
```

Core of the blake2b compression function, takes a workspace v and scrambles four positions in it, specified by the four integers a,b,c,d, using the additional data contained in x and y.

**Parameters**

| | |
|---|---|
| *v* | workspace for the scrambling, an array of 16 uint64_t |
| *a* | the first position to scramble in v |
| *b* | the second position to scramble in v |
| *c* | the third position to scramble in v |
| *d* | the fourth position to scramble in v |
| *x* | additional data for the scrambling |
| *y* | additional data for the scrambling |

#### 5.9.3.3 blake2b()

```
void blake2b (
            uint8_t * digest,
            size_t digest_size,
            uint8_t * data,
            uint64_t data_size )
```

Simplified version of the blake2b hash function. It divides the input data into blocks and then proceeds to repeatedly compress them, xoring the result in h, an array of 64 bytes, then outputs the required amount of bytes from it. Unlike the original blake2b, this version does not accept a key, since it is unnecessary for Argon2 computation.

**Parameters**

| | |
|---|---|
| *digest* | an array to store the digest |
| *digest_size* | the number of required output bytes |
| *data* | the input data to compress |
| *data_size* | the size of said data, expressed in number of bytes |

## 5.10 Blake2b.h File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <stdlib.h>
```

Include dependency graph for Blake2b.h: This graph shows which files directly or indirectly include this file:

### Macros

- #define ERROR(msg) {puts((char∗)msg); exit(1);}
- #define ROT_SHIFT(array, offset) (((array) >> (offset)) ^ ((array) << (64 - (offset))))

### Functions

- void blake2b (uint8_t ∗digest, size_t digest_size, uint8_t ∗data, uint64_t data_size)

### 5.10.1 Detailed Description

Interface for the blake2b hash function

### 5.10.2 Macro Definition Documentation

#### 5.10.2.1 ERROR

```
#define ERROR(
             msg ) {puts((char*)msg); exit(1);}
```

An utility function, used to print an error to std out and terminate the program

#### 5.10.2.2 ROT_SHIFT

```
#define ROT_SHIFT(
             array,
             offset ) (((array) >> (offset)) ^ ((array) << (64 - (offset))))
```

An utility function, used to apply a rotational right shift of 'offset' positions to an array

### 5.10.3 Function Documentation

#### 5.10.3.1 blake2b()

```
void blake2b (
            uint8_t * digest,
            size_t digest_size,
            uint8_t * data,
            uint64_t data_size )
```

Simplified version of the blake2b hash function. It divides the input data into blocks and then proceeds to repeatedly compress them, xoring the result in h, an array of 64 bytes, then outputs the required amount of bytes from it. Unlike the original blake2b, this version does not accept a key, since it is unnecessary for Argon2 computation.

**Parameters**

| | |
|---|---|
| *digest* | an array to store the digest |
| *digest_size* | the number of required output bytes |
| *data* | the input data to compress |
| *data_size* | the size of said data, expressed in number of bytes |

## 5.11 README.md File Reference

## 5.12 test.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include "Argon2_body.h"
```

Include dependency graph for test.c:

### Functions

- int main ()

### Variables

- uint8_t A2i_tags [3][32]
- uint8_t A2d_tags [3][32]
- uint8_t A2id_tags [3][32]
- uint8_t memory_test [32] = {0x1f, 0x92, 0x78, 0x12, 0xd1, 0x19, 0x30, 0x74, 0x2f, 0x9a, 0x54, 0x5b, 0x7a, 0xce, 0xaf, 0xc4, 0x64, 0xb8, 0x43, 0x9d, 0xb9, 0x06, 0x1b, 0x01, 0x28, 0x82, 0xeb, 0x87, 0xe8, 0xd9, 0x3a, 0xa9}

### 5.12.1 Detailed Description

Test for the correctness of our implementation, runs a test for different parameters, considered critical:
(°) Simple test with p = 4 and m = 32;
(°) Test for p = 5 and m = 2601, so that p does not divide m, and the segment length is enough to require a second pseudo-random generation in Argon2i;
(°) Test for p = 4 and m = 65536, to test fairly large amounts of memory used;
(°) A final test using 4 GiB of memory, in order to test the memory handling.
Remark: there are no available test vectors for the Argon2ds type using version 1.9, since it has not been implemented.
The fixed parameters for the test are:
Iterations: 3, Tag length: 32 bytes Password[32]: 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 Salt[16]: 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 Secret[8]: 03 03 03 03 03 03 03 03 Associated data[12]: 04 04 04 04 04 04 04 04 04 04 04 04

Test main, initializes parametrs for the tests, launches them and prints the resulting Tag and whether the test has been successful.

### 5.12.2 Function Documentation

#### 5.12.2.1 main()

```
int main ( )
```

### 5.12.3 Variable Documentation

#### 5.12.3.1 A2d_tags

```
A2d_tags
```

**Initial value:**

```
= {

    {0x51, 0x2b, 0x39, 0x1b, 0x6f, 0x11, 0x62, 0x97, 0x53, 0x71, 0xd3, 0x09, 0x19, 0x73, 0x42, 0x94, 0xf8,
      0x68, 0xe3, 0xbe, 0x39, 0x84, 0xf3, 0xc1, 0xa1, 0x3a, 0x4d, 0xb9, 0xfa, 0xbe, 0x4a, 0xcb},
    {0xf3, 0x17, 0x84, 0x0d, 0x31, 0x43, 0xfe, 0xdb, 0x5c, 0x91, 0x6c, 0xc5, 0x90, 0x51, 0x66, 0x60, 0xf6,
      0x83, 0x08, 0x46, 0x38, 0x32, 0xe6, 0x20, 0xc3, 0xb0, 0x1e, 0x57, 0x7e, 0x27, 0x6b, 0xdd},
    {0xba, 0x79, 0xe5, 0x47, 0x42, 0x5e, 0xb4, 0x7b, 0xd7, 0x7c, 0x22, 0x51, 0xd4, 0xdf, 0x02, 0x9c, 0x32,
      0xc5, 0xc5, 0xf7, 0x76, 0x92, 0xc7, 0x84, 0xeb, 0xc1, 0x1b, 0x8f, 0xe7, 0x8a, 0xb5, 0x09}

}
```

The results for the Argon2d tests, computed with the official version.

#### 5.12.3.2 A2i_tags

```
A2i_tags
```

**Initial value:**

```
= {

    {0xc8, 0x14, 0xd9, 0xd1, 0xdc, 0x7f, 0x37, 0xaa, 0x13, 0xf0, 0xd7, 0x7f, 0x24, 0x94, 0xbd, 0xa1, 0xc8,
      0xde, 0x6b, 0x01, 0x6d, 0xd3, 0x88, 0xd2, 0x99, 0x52, 0xa4, 0xc4, 0x67, 0x2b, 0x6c, 0xe8},
    {0xdc, 0x22, 0x95, 0xaf, 0x5f, 0x5e, 0x9e, 0xce, 0x87, 0x25, 0x1c, 0x42, 0x72, 0x35, 0xbc, 0x25, 0xfc,
      0x9f, 0x60, 0x4a, 0xbd, 0x77, 0xdc, 0x4a, 0x70, 0x57, 0x60, 0xa2, 0x93, 0xfb, 0x4b, 0xed},
    {0x80, 0xf2, 0x82, 0x95, 0xc9, 0x74, 0xad, 0x00, 0x6d, 0x74, 0x97, 0x01, 0xdc, 0x38, 0xf1, 0x62, 0xb0,
      0xee, 0x11, 0x8b, 0xd7, 0xb9, 0x07, 0x43, 0x37, 0x9e, 0xb7, 0xc0, 0x78, 0xce, 0x95, 0xdc}


}
```

The results for the Argon2i tests, computed with the official version.

#### 5.12.3.3 A2id_tags

```
A2id_tags
```

**Initial value:**

```
= {

    {0x0d, 0x64, 0x0d, 0xf5, 0x8d, 0x78, 0x76, 0x6c, 0x08, 0xc0, 0x37, 0xa3, 0x4a, 0x8b, 0x53, 0xc9, 0xd0,
      0x1e, 0xf0, 0x45, 0x2d, 0x75, 0xb6, 0x5e, 0xb5, 0x25, 0x20, 0xe9, 0x6b, 0x01, 0xe6, 0x59},
    {0xe8, 0x00, 0xa4, 0x0d, 0x4f, 0xde, 0x5b, 0x25, 0x53, 0x5e, 0x5c, 0xf7, 0x96, 0x05, 0xcc, 0x5c, 0x04,
      0x3c, 0x4c, 0xc2, 0x69, 0x34, 0x2d, 0x80, 0x54, 0x07, 0x09, 0xd1, 0x74, 0xc1, 0xf9, 0xa5},
    {0x47, 0xa6, 0x77, 0xeb, 0xc6, 0x84, 0x12, 0xb8, 0xe1, 0x79, 0x49, 0xc9, 0x21, 0xfd, 0xca, 0x06, 0x53,
      0x52, 0x11, 0x4f, 0xfb, 0x2a, 0x09, 0xee, 0xbe, 0x61, 0xad, 0xc4, 0x51, 0xee, 0xcf, 0x1e}

}
```

The results for the Argon2id tests, computed with the official version.

#### 5.12.3.4 memory_test

```
memory_test = {0x1f, 0x92, 0x78, 0x12, 0xd1, 0x19, 0x30, 0x74, 0x2f, 0x9a, 0x54, 0x5b, 0x7a,
0xce, 0xaf, 0xc4, 0x64, 0xb8, 0x43, 0x9d, 0xb9, 0x06, 0x1b, 0x01, 0x28, 0x82, 0xeb, 0x87,
0xe8, 0xd9, 0x3a, 0xa9}
```

The result for the test using 4 GiB of memory

# Index