

Ordo  
2.7.0

Generated by Doxygen 1.8.6

Thu Jan 30 2014 01:11:08



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>README</b>	<b>3</b>
<b>3</b>	<b>Data Structure Index</b>	<b>5</b>
3.1	Data Structures . . . . .	5
<b>4</b>	<b>File Index</b>	<b>7</b>
4.1	File List . . . . .	7
<b>5</b>	<b>Data Structure Documentation</b>	<b>9</b>
5.1	AES_PARAMS Struct Reference . . . . .	9
5.1.1	Detailed Description . . . . .	9
5.1.2	Field Documentation . . . . .	9
5.1.2.1	rounds . . . . .	9
5.2	CBC_PARAMS Struct Reference . . . . .	9
5.2.1	Detailed Description . . . . .	10
5.2.2	Field Documentation . . . . .	10
5.2.2.1	padding . . . . .	10
5.3	ECB_PARAMS Struct Reference . . . . .	10
5.3.1	Detailed Description . . . . .	10
5.3.2	Field Documentation . . . . .	10
5.3.2.1	padding . . . . .	10
5.4	ORDO_VERSION Struct Reference . . . . .	10
5.4.1	Detailed Description . . . . .	11
5.5	RC4_PARAMS Struct Reference . . . . .	11
5.5.1	Detailed Description . . . . .	11
5.5.2	Field Documentation . . . . .	11
5.5.2.1	drop . . . . .	11
5.6	SKEIN256_PARAMS Struct Reference . . . . .	12
5.6.1	Detailed Description . . . . .	12
5.6.2	Field Documentation . . . . .	12
5.6.2.1	out_len . . . . .	12

5.7	THREEFISH256_PARAMS Struct Reference . . . . .	13
5.7.1	Detailed Description . . . . .	13
<b>6</b>	<b>File Documentation</b>	<b>15</b>
6.1	/home/tom/Projects/github/Ordo/include/ordo.h File Reference . . . . .	15
6.1.1	Detailed Description . . . . .	16
6.1.2	Function Documentation . . . . .	16
6.1.2.1	ordo_allocator . . . . .	16
6.1.2.2	ordo_enc_block . . . . .	16
6.1.2.3	ordo_enc_stream . . . . .	17
6.1.2.4	ordo_digest . . . . .	17
6.1.2.5	ordo_hmac . . . . .	18
6.2	/home/tom/Projects/github/Ordo/include/ordo/auth/hmac.h File Reference . . . . .	18
6.2.1	Detailed Description . . . . .	20
6.2.2	Function Documentation . . . . .	20
6.2.2.1	hmac_alloc . . . . .	20
6.2.2.2	hmac_init . . . . .	20
6.2.2.3	hmac_update . . . . .	21
6.2.2.4	hmac_final . . . . .	22
6.2.2.5	hmac_free . . . . .	22
6.2.2.6	hmac_copy . . . . .	22
6.3	/home/tom/Projects/github/Ordo/include/ordo/common/error.h File Reference . . . . .	23
6.3.1	Detailed Description . . . . .	23
6.3.2	Enumeration Type Documentation . . . . .	23
6.3.2.1	ORDO_ERROR . . . . .	23
6.3.3	Function Documentation . . . . .	24
6.3.3.1	ordo_error_msg . . . . .	24
6.4	/home/tom/Projects/github/Ordo/include/ordo/common/interface.h File Reference . . . . .	25
6.4.1	Detailed Description . . . . .	25
6.5	/home/tom/Projects/github/Ordo/include/ordo/common/query.h File Reference . . . . .	25
6.5.1	Detailed Description . . . . .	26
6.5.2	Enumeration Type Documentation . . . . .	26
6.5.2.1	ORDO_QUERY . . . . .	26
6.6	/home/tom/Projects/github/Ordo/include/ordo/common/version.h File Reference . . . . .	27
6.6.1	Detailed Description . . . . .	27
6.7	/home/tom/Projects/github/Ordo/include/ordo/digest/digest.h File Reference . . . . .	27
6.7.1	Detailed Description . . . . .	29
6.7.2	Function Documentation . . . . .	30
6.7.2.1	digest_alloc . . . . .	30
6.7.2.2	digest_init . . . . .	30

6.7.2.3	<a href="#">digest_update</a>	30
6.7.2.4	<a href="#">digest_final</a>	31
6.7.2.5	<a href="#">digest_free</a>	31
6.7.2.6	<a href="#">digest_copy</a>	31
6.7.2.7	<a href="#">digest_length</a>	32
6.8	<a href="#">/home/tom/Projects/github/Ordo/include/ordo/enc/enc_block.h File Reference</a>	32
6.8.1	<a href="#">Detailed Description</a>	33
6.8.2	<a href="#">Function Documentation</a>	33
6.8.2.1	<a href="#">enc_block_alloc</a>	34
6.8.2.2	<a href="#">enc_block_init</a>	35
6.8.2.3	<a href="#">enc_block_update</a>	35
6.8.2.4	<a href="#">enc_block_final</a>	36
6.8.2.5	<a href="#">enc_block_free</a>	37
6.8.2.6	<a href="#">enc_block_copy</a>	37
6.8.2.7	<a href="#">enc_block_key_len</a>	37
6.8.2.8	<a href="#">enc_block_iv_len</a>	38
6.9	<a href="#">/home/tom/Projects/github/Ordo/include/ordo/enc/enc_stream.h File Reference</a>	39
6.9.1	<a href="#">Detailed Description</a>	40
6.9.2	<a href="#">Function Documentation</a>	40
6.9.2.1	<a href="#">enc_stream_alloc</a>	40
6.9.2.2	<a href="#">enc_stream_init</a>	41
6.9.2.3	<a href="#">enc_stream_update</a>	42
6.9.2.4	<a href="#">enc_stream_final</a>	42
6.9.2.5	<a href="#">enc_stream_free</a>	42
6.9.2.6	<a href="#">enc_stream_copy</a>	42
6.9.2.7	<a href="#">enc_stream_key_len</a>	43
6.10	<a href="#">/home/tom/Projects/github/Ordo/include/ordo/internal/alg.h File Reference</a>	44
6.10.1	<a href="#">Detailed Description</a>	44
6.10.2	<a href="#">Macro Definition Documentation</a>	44
6.10.2.1	<a href="#">bits</a>	44
6.10.2.2	<a href="#">bytes</a>	44
6.10.2.3	<a href="#">offset</a>	45
6.10.3	<a href="#">Function Documentation</a>	46
6.10.3.1	<a href="#">pad_check</a>	46
6.10.3.2	<a href="#">xor_buffer</a>	46
6.10.3.3	<a href="#">inc_buffer</a>	46
6.11	<a href="#">/home/tom/Projects/github/Ordo/include/ordo/internal/implementation.h File Reference</a>	47
6.11.1	<a href="#">Detailed Description</a>	47
6.12	<a href="#">/home/tom/Projects/github/Ordo/include/ordo/internal/mem.h File Reference</a>	47
6.12.1	<a href="#">Detailed Description</a>	47

6.12.2	Function Documentation	47
6.12.2.1	mem_alloc	47
6.12.2.2	mem_free	48
6.12.2.3	mem_erase	48
6.13	/home/tom/Projects/github/Ordo/include/ordo/internal/sys.h File Reference	48
6.13.1	Detailed Description	48
6.14	/home/tom/Projects/github/Ordo/include/ordo/kdf/pbkdf2.h File Reference	48
6.14.1	Detailed Description	50
6.14.2	Function Documentation	50
6.14.2.1	pbkdf2	50
6.15	/home/tom/Projects/github/Ordo/include/ordo/misc/endianness.h File Reference	50
6.15.1	Detailed Description	50
6.16	/home/tom/Projects/github/Ordo/include/ordo/misc/os_random.h File Reference	51
6.16.1	Detailed Description	51
6.16.2	Function Documentation	51
6.16.2.1	os_random	51
6.17	/home/tom/Projects/github/Ordo/include/ordo/primitives/block_ciphers.h File Reference	52
6.17.1	Detailed Description	53
6.17.2	Function Documentation	53
6.17.2.1	block_cipher_name	53
6.17.2.2	block_cipher_count	53
6.17.2.3	block_cipher_by_name	53
6.17.2.4	block_cipher_by_index	54
6.17.2.5	block_cipher_alloc	54
6.17.2.6	block_cipher_init	54
6.17.2.7	block_cipher_forward	55
6.17.2.8	block_cipher_inverse	56
6.17.2.9	block_cipher_final	56
6.17.2.10	block_cipher_free	56
6.17.2.11	block_cipher_copy	56
6.17.2.12	block_cipher_query	57
6.18	/home/tom/Projects/github/Ordo/include/ordo/primitives/block_ciphers/aes.h File Reference	57
6.18.1	Detailed Description	58
6.18.2	Function Documentation	58
6.18.2.1	aes_alloc	58
6.18.2.2	aes_init	58
6.18.2.3	aes_forward	58
6.18.2.4	aes_inverse	58
6.18.2.5	aes_final	59
6.18.2.6	aes_free	59

6.18.2.7	<a href="#">aes_copy</a>	59
6.18.2.8	<a href="#">aes_query</a>	59
6.19	<a href="#">/home/tom/Projects/github/Ordo/include/ordo/primitives/block_ciphers/block_params.h File Reference</a>	59
6.19.1	Detailed Description	60
6.20	<a href="#">/home/tom/Projects/github/Ordo/include/ordo/primitives/block_ciphers/nullcipher.h File Reference</a>	60
6.20.1	Detailed Description	61
6.20.2	Function Documentation	61
6.20.2.1	<a href="#">nullcipher_alloc</a>	61
6.20.2.2	<a href="#">nullcipher_init</a>	61
6.20.2.3	<a href="#">nullcipher_forward</a>	61
6.20.2.4	<a href="#">nullcipher_inverse</a>	61
6.20.2.5	<a href="#">nullcipher_final</a>	61
6.20.2.6	<a href="#">nullcipher_free</a>	61
6.20.2.7	<a href="#">nullcipher_copy</a>	61
6.20.2.8	<a href="#">nullcipher_query</a>	62
6.21	<a href="#">/home/tom/Projects/github/Ordo/include/ordo/primitives/block_ciphers/threefish256.h File Reference</a>	62
6.21.1	Detailed Description	62
6.21.2	Function Documentation	63
6.21.2.1	<a href="#">threefish256_alloc</a>	63
6.21.2.2	<a href="#">threefish256_init</a>	63
6.21.2.3	<a href="#">threefish256_forward</a>	63
6.21.2.4	<a href="#">threefish256_inverse</a>	63
6.21.2.5	<a href="#">threefish256_final</a>	63
6.21.2.6	<a href="#">threefish256_free</a>	63
6.21.2.7	<a href="#">threefish256_copy</a>	63
6.21.2.8	<a href="#">threefish256_query</a>	63
6.22	<a href="#">/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes.h File Reference</a>	64
6.22.1	Detailed Description	66
6.22.2	Function Documentation	66
6.22.2.1	<a href="#">block_mode_name</a>	66
6.22.2.2	<a href="#">block_mode_count</a>	66
6.22.2.3	<a href="#">block_mode_by_name</a>	66
6.22.2.4	<a href="#">block_mode_by_index</a>	67
6.22.2.5	<a href="#">block_mode_alloc</a>	67
6.22.2.6	<a href="#">block_mode_init</a>	67
6.22.2.7	<a href="#">block_mode_update</a>	68
6.22.2.8	<a href="#">block_mode_final</a>	68
6.22.2.9	<a href="#">block_mode_free</a>	69
6.22.2.10	<a href="#">block_mode_copy</a>	70
6.22.2.11	<a href="#">block_mode_query</a>	70

6.23	/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/cbc.h File Reference . . . . .	70
6.23.1	Detailed Description . . . . .	71
6.23.2	Function Documentation . . . . .	71
6.23.2.1	cbc_alloc . . . . .	72
6.23.2.2	cbc_init . . . . .	72
6.23.2.3	cbc_update . . . . .	72
6.23.2.4	cbc_final . . . . .	72
6.23.2.5	cbc_free . . . . .	72
6.23.2.6	cbc_copy . . . . .	72
6.23.2.7	cbc_query . . . . .	72
6.24	/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/cfb.h File Reference . . . . .	72
6.24.1	Detailed Description . . . . .	73
6.24.2	Function Documentation . . . . .	73
6.24.2.1	cfb_alloc . . . . .	73
6.24.2.2	cfb_init . . . . .	74
6.24.2.3	cfb_update . . . . .	74
6.24.2.4	cfb_final . . . . .	74
6.24.2.5	cfb_free . . . . .	74
6.24.2.6	cfb_copy . . . . .	74
6.24.2.7	cfb_query . . . . .	74
6.25	/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/ctr.h File Reference . . . . .	74
6.25.1	Detailed Description . . . . .	75
6.25.2	Function Documentation . . . . .	75
6.25.2.1	ctr_alloc . . . . .	76
6.25.2.2	ctr_init . . . . .	76
6.25.2.3	ctr_update . . . . .	76
6.25.2.4	ctr_final . . . . .	76
6.25.2.5	ctr_free . . . . .	76
6.25.2.6	ctr_copy . . . . .	76
6.25.2.7	ctr_query . . . . .	76
6.26	/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/ecb.h File Reference . . . . .	76
6.26.1	Detailed Description . . . . .	77
6.26.2	Function Documentation . . . . .	77
6.26.2.1	ecb_alloc . . . . .	77
6.26.2.2	ecb_init . . . . .	78
6.26.2.3	ecb_update . . . . .	78
6.26.2.4	ecb_final . . . . .	78
6.26.2.5	ecb_free . . . . .	78
6.26.2.6	ecb_copy . . . . .	78
6.26.2.7	ecb_query . . . . .	78



6.27	/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/mode_params.h File Reference	78
6.27.1	Detailed Description	79
6.28	/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/ofb.h File Reference	79
6.28.1	Detailed Description	80
6.28.2	Function Documentation	80
6.28.2.1	ofb_alloc	80
6.28.2.2	ofb_init	80
6.28.2.3	ofb_update	80
6.28.2.4	ofb_final	80
6.28.2.5	ofb_free	81
6.28.2.6	ofb_copy	81
6.28.2.7	ofb_query	81
6.29	/home/tom/Projects/github/Ordo/include/ordo/primitives/hash_functions.h File Reference	81
6.29.1	Detailed Description	83
6.29.2	Function Documentation	83
6.29.2.1	hash_function_name	83
6.29.2.2	hash_function_count	83
6.29.2.3	hash_function_by_name	84
6.29.2.4	hash_function_by_index	85
6.29.2.5	hash_function_alloc	85
6.29.2.6	hash_function_init	85
6.29.2.7	hash_function_update	85
6.29.2.8	hash_function_final	86
6.29.2.9	hash_function_free	86
6.29.2.10	hash_function_copy	86
6.29.2.11	hash_function_query	86
6.30	/home/tom/Projects/github/Ordo/include/ordo/primitives/hash_functions/hash_params.h File Reference	87
6.30.1	Detailed Description	87
6.31	/home/tom/Projects/github/Ordo/include/ordo/primitives/hash_functions/md5.h File Reference	88
6.31.1	Detailed Description	88
6.31.2	Function Documentation	88
6.31.2.1	md5_alloc	88
6.31.2.2	md5_init	88
6.31.2.3	md5_update	89
6.31.2.4	md5_final	89
6.31.2.5	md5_free	89
6.31.2.6	md5_copy	89
6.31.2.7	md5_query	89
6.32	/home/tom/Projects/github/Ordo/include/ordo/primitives/hash_functions/sha256.h File Reference	89

6.32.1	Detailed Description	90
6.32.2	Function Documentation	90
6.32.2.1	sha256_alloc	90
6.32.2.2	sha256_init	90
6.32.2.3	sha256_update	91
6.32.2.4	sha256_final	91
6.32.2.5	sha256_free	91
6.32.2.6	sha256_copy	91
6.32.2.7	sha256_query	91
6.33	/home/tom/Projects/github/Ordo/include/ordo/primitives/hash_functions/skein256.h File Reference	92
6.33.1	Detailed Description	92
6.33.2	Function Documentation	93
6.33.2.1	skein256_alloc	93
6.33.2.2	skein256_init	93
6.33.2.3	skein256_update	93
6.33.2.4	skein256_final	93
6.33.2.5	skein256_free	93
6.33.2.6	skein256_copy	93
6.33.2.7	skein256_query	93
6.34	/home/tom/Projects/github/Ordo/include/ordo/primitives/stream_ciphers.h File Reference	94
6.34.1	Detailed Description	95
6.34.2	Function Documentation	95
6.34.2.1	stream_cipher_name	95
6.34.2.2	stream_cipher_count	95
6.34.2.3	stream_cipher_by_name	95
6.34.2.4	stream_cipher_by_index	96
6.34.2.5	stream_cipher_alloc	97
6.34.2.6	stream_cipher_init	97
6.34.2.7	stream_cipher_update	97
6.34.2.8	stream_cipher_final	98
6.34.2.9	stream_cipher_free	98
6.34.2.10	stream_cipher_copy	98
6.34.2.11	stream_cipher_query	98
6.35	/home/tom/Projects/github/Ordo/include/ordo/primitives/stream_ciphers/rc4.h File Reference	99
6.35.1	Detailed Description	99
6.35.2	Function Documentation	99
6.35.2.1	rc4_alloc	99
6.35.2.2	rc4_init	100
6.35.2.3	rc4_update	100
6.35.2.4	rc4_final	100

---

6.35.2.5	<a href="#">rc4_free</a>	100
6.35.2.6	<a href="#">rc4_copy</a>	100
6.35.2.7	<a href="#">rc4_query</a>	100
6.36	<a href="#">/home/tom/Projects/github/Ordo/include/ordo/primitives/stream_ciphers/stream_params.h File Reference</a>	101
6.36.1	<a href="#">Detailed Description</a>	101
<b>Index</b>		<b>102</b>



# Chapter 1

## Main Page

### Symmetric Cryptography Library

This is the github repository for Ordo, a minimalist cryptography library with an emphasis on symmetric cryptography, which strives to meet high performance, portability, and security standards, while remaining modular in design to facilitate adding new features and maintaining existing ones. The library is written in standard C with system-specific features, but some sections are assembly-optimized for efficiency. Note that while the library is technically usable at this point, it is still very much a work in progress and mustn't be deployed in security-sensitive applications.

### Status

! [Build Status] (<https://travis-ci.org/TomCrypto/Ordo.png?branch=master>)

What's new in 2.7.0:

- the endianness API has been made public, from internal
- all library initialization is now implicit, no more `ordo_init`
- the version code has been improved

### Feature Map

This table doesn't include every single feature but gives a high level overview of what is available so far:

Block Ciphers	Stream Ciphers	Hash Functions	Modes	Authenticat-ion	Key Derivation	Misc
AES	RC4	MD5	ECB	HMAC	PBKDF2	CSPRNG
Threefish-256	-	SHA-256	CBC	-	-	-
-	-	Skein-256	OFB	-	-	-
-	-	-	CFB	-	-	-
-	-	-	CTR	-	-	-

### Documentation

Ordo is documented for Doxygen, and you can automatically generate all documentation by using the `doc build` target, if deemed available on your system (you will need `doxygen`, and `pdflatex` with a working TeX environment for the LaTeX output). The HTML documentation will be generated in `doc/html`, and the LaTeX documentation will be generated in `doc/latex`, which you can then typeset using the generated makefile.

You can also access a recent version of the documentation online through the [project page](#).

## How To Build

We support recent versions of MSVC, GCC, MinGW, and Clang. Other compilers are not officially supported. The build system used is CMake, which has a few configuration options to tweak the library according to your needs. A `build` folder is provided for you to point CMake to.

- `LTO`: use link-time optimization, this should be enabled for optimal performance.
- `ARCH`: the architecture to use, pick the one most appropriate for your hardware.

Note the system is autodetected and automatically included in the build. Additional options, such as the use of special hardware instructions, may become available once an architecture is selected, if they are supported. Link-time optimization may not be available on older compilers (it will let you know).

If you are not using the `cmake-gui` utility, the command-line options to configure the library are:

```
cd build && cmake .. [-DARCH=arch] [[-DFEATURE=on] ...] [-DLTO=off]
```

For instance, a typical configuration for `x86_64` machines with the AES-NI instructions could be:

```
cd build && cmake .. -DARCH=amd64 -DAES_NI=on
```

The test driver is in the `test` folder, the sample programs are in the `samples` folder.

## Assembly Support

We use the NASM assembler for our assembly files. For Linux and other Unix-based operating systems this should work out of the box after installing the assembler. For MSVC on Windows using the Visual Studio generators, custom build rules have been set up to autodetect NASM and get it to automatically compile assembly files, but they have not been tested (and may not necessarily work) for all versions of Visual Studio.

## Static Linking

If you wish to link statically to the library, please define the `ORDO_STATIC_LIB` preprocessor token in your project so that the Ordo headers can configure themselves accordingly (otherwise, they will assume you are linking to a shared library, which may raise some unwelcome compiler warnings as well as forbidding access to the internal headers).

## Compatibility

The library will run everywhere a C99 compiler (with `stdint.h` and a couple other C99 features) is available, however system-dependent modules will not be available without an implementation for these platforms. For better performance, specialized algorithm implementations may be available for your system and processor architecture, and are easy to integrate once written.

## Conclusion

Of course, do not use Ordo for anything other than testing or contributing for now! It can only be used once it has been completed and extensively checked (and even then, there may still be flaws and bugs, as in any other software).

## Chapter 2

# README

This directory stores system implementations which are applicable to multiple systems without modifications. Systems, or system groups, in this directory are not intended to be directly added to the build, but are to be symlinked as needed by the proper system implementations. This mechanism greatly reduces code duplication and improves maintainability.

As an example, much of the `unix` directory is referenced from `linux`, `freebsd`, `openbsd`, `netbsd`, and `darwin`, as they usually share the same ABI and have many system features in common (such as `/dev/urandom`). An exception is the `endianness.c` source file which differs slightly across those systems.





## Chapter 3

# Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">AES_PARAMS</a>	
AES block cipher parameters . . . . .	9
<a href="#">CBC_PARAMS</a>	
CBC parameters . . . . .	9
<a href="#">ECB_PARAMS</a>	
ECB parameters . . . . .	10
<a href="#">ORDO_VERSION</a>	
Library version information . . . . .	10
<a href="#">RC4_PARAMS</a>	
RC4 stream cipher parameters . . . . .	11
<a href="#">SKEIN256_PARAMS</a>	
Skein-256 hash function parameters . . . . .	12
<a href="#">THREEFISH256_PARAMS</a>	
Threefish-256 block cipher parameters . . . . .	13



## Chapter 4

# File Index

### 4.1 File List

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

/home/tom/Projects/github/Ordo/include/ <a href="#">ordo.h</a>	
Wrapper . . . . .	15
/home/tom/Projects/github/Ordo/include/ordo/auth/ <a href="#">hmac.h</a>	
Module . . . . .	18
/home/tom/Projects/github/Ordo/include/ordo/common/ <a href="#">error.h</a>	
Utility . . . . .	23
/home/tom/Projects/github/Ordo/include/ordo/common/ <a href="#">interface.h</a>	
API . . . . .	25
/home/tom/Projects/github/Ordo/include/ordo/common/ <a href="#">query.h</a>	
Utility . . . . .	25
/home/tom/Projects/github/Ordo/include/ordo/common/ <a href="#">version.h</a>	
Utility . . . . .	27
/home/tom/Projects/github/Ordo/include/ordo/digest/ <a href="#">digest.h</a>	
Module . . . . .	27
/home/tom/Projects/github/Ordo/include/ordo/enc/ <a href="#">enc_block.h</a>	
Module . . . . .	32
/home/tom/Projects/github/Ordo/include/ordo/enc/ <a href="#">enc_stream.h</a>	
Module . . . . .	39
/home/tom/Projects/github/Ordo/include/ordo/internal/ <a href="#">alg.h</a>	
<b>Internal</b> , Utility . . . . .	44
/home/tom/Projects/github/Ordo/include/ordo/internal/ <a href="#">implementation.h</a>	
<b>Internal</b> , API . . . . .	47
/home/tom/Projects/github/Ordo/include/ordo/internal/ <a href="#">mem.h</a>	
<b>Internal</b> , Utility . . . . .	47
/home/tom/Projects/github/Ordo/include/ordo/internal/ <a href="#">sys.h</a>	
<b>Internal</b> , Utility . . . . .	48
/home/tom/Projects/github/Ordo/include/ordo/kdf/ <a href="#">pbkdf2.h</a>	
Module . . . . .	48
/home/tom/Projects/github/Ordo/include/ordo/misc/ <a href="#">endianness.h</a>	
Utility . . . . .	50
/home/tom/Projects/github/Ordo/include/ordo/misc/ <a href="#">os_random.h</a>	
Module . . . . .	51
/home/tom/Projects/github/Ordo/include/ordo/primitives/ <a href="#">block_ciphers.h</a>	
Abstraction Layer . . . . .	52
/home/tom/Projects/github/Ordo/include/ordo/primitives/ <a href="#">block_modes.h</a>	
Abstraction Layer . . . . .	64
/home/tom/Projects/github/Ordo/include/ordo/primitives/ <a href="#">hash_functions.h</a>	
Abstraction Layer . . . . .	81

/home/tom/Projects/github/Ordo/include/ordo/primitives/stream_ciphers.h	
Abstraction Layer	94
/home/tom/Projects/github/Ordo/include/ordo/primitives/block_ciphers/aes.h	
Primitive	57
/home/tom/Projects/github/Ordo/include/ordo/primitives/block_ciphers/block_params.h	
Primitive Parameters	59
/home/tom/Projects/github/Ordo/include/ordo/primitives/block_ciphers/nullcipher.h	
Primitive	60
/home/tom/Projects/github/Ordo/include/ordo/primitives/block_ciphers/threefish256.h	
Primitive	62
/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/cbc.h	
Primitive	70
/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/cfb.h	
Primitive	72
/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/ctr.h	
Primitive	74
/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/ecb.h	
Primitive	76
/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/mode_params.h	
Primitive Parameters	78
/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/ofb.h	
Primitive	79
/home/tom/Projects/github/Ordo/include/ordo/primitives/hash_functions/hash_params.h	
Primitive Parameters	87
/home/tom/Projects/github/Ordo/include/ordo/primitives/hash_functions/md5.h	
Primitive	88
/home/tom/Projects/github/Ordo/include/ordo/primitives/hash_functions/sha256.h	
Primitive	89
/home/tom/Projects/github/Ordo/include/ordo/primitives/hash_functions/skein256.h	
Primitive	92
/home/tom/Projects/github/Ordo/include/ordo/primitives/stream_ciphers/rc4.h	
Primitive	99
/home/tom/Projects/github/Ordo/include/ordo/primitives/stream_ciphers/stream_params.h	
Primitive Parameters	101

## Chapter 5

# Data Structure Documentation

### 5.1 AES\_PARAMS Struct Reference

AES block cipher parameters.

```
#include <block_params.h>
```

#### Data Fields

- `size_t` [rounds](#)

#### 5.1.1 Detailed Description

AES block cipher parameters.

#### 5.1.2 Field Documentation

##### 5.1.2.1 `size_t` rounds

The number of rounds to use.

#### Remarks

The defaults are 10 for a 128-bit key, 12 for a 192-bit key, 14 for a 256-bit key, and are standardized. It is **strongly** discouraged to lower the number of rounds below the defaults.

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

- `/home/tom/Projects/github/Ordo/include/ordo/primitives/block_ciphers/block\_params.h`

### 5.2 CBC\_PARAMS Struct Reference

CBC parameters.

```
#include <mode_params.h>
```

#### Data Fields

- `size_t` [padding](#)

### 5.2.1 Detailed Description

CBC parameters.

### 5.2.2 Field Documentation

#### 5.2.2.1 `size_t` padding

Whether padding should be used.

#### Remarks

Set to 0 to disable padding, and 1 to enable it - only the least significant bit is used, all other bits are ignored. Padding is enabled by default if parameters are not used.

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

- `/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/mode\_params.h`

## 5.3 ECB\_PARAMS Struct Reference

ECB parameters.

```
#include <mode_params.h>
```

### Data Fields

- `size_t` [padding](#)

### 5.3.1 Detailed Description

ECB parameters.

### 5.3.2 Field Documentation

#### 5.3.2.1 `size_t` padding

Whether padding should be used.

#### Remarks

Set to 0 to disable padding, and 1 to enable it - only the least significant bit is used, all other bits are ignored. Padding is enabled by default if parameters are not used.

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

- `/home/tom/Projects/github/Ordo/include/ordo/primitives/block_modes/mode\_params.h`

## 5.4 ORDO\_VERSION Struct Reference

Library version information.

```
#include <version.h>
```

## Data Fields

- unsigned int [id](#)  
*The version as an integer of the form XXYYZZ, e.g. 30242 == 3.2.42.*
- const char \* [version](#)  
*The version e.g. "2.7.0".*
- const char \* [system](#)  
*The target system e.g. "linux".*
- const char \* [arch](#)  
*The target architecture e.g. "amd64".*
- const char \* [build](#)  
*A string which contains version, system and architecture.*
- const char \*const \* [features](#)  
*A null-terminated list of targeted features.*
- const char \* [feature\\_list](#)  
*The list of features, as a space-separated string.*

### 5.4.1 Detailed Description

Library version information.

Contains version information for the library.

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

- `/home/tom/Projects/github/Ordo/include/ordo/common/version.h`

## 5.5 RC4\_PARAMS Struct Reference

RC4 stream cipher parameters.

```
#include <stream_params.h>
```

## Data Fields

- size\_t [drop](#)

### 5.5.1 Detailed Description

RC4 stream cipher parameters.

### 5.5.2 Field Documentation

#### 5.5.2.1 size\_t drop

The number of keystream bytes to drop prior to encryption.

#### Remarks

Setting this implements the given RC4-drop variant.

If this [RC4\\_PARAMS](#) structure is **not** passed to the RC4 stream cipher primitive, the default drop amount is 2048.

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

- [/home/tom/Projects/github/Ordo/include/ordo/primitives/stream\\_ciphers/stream\\_params.h](#)

## 5.6 SKEIN256\_PARAMS Struct Reference

Skein-256 hash function parameters.

```
#include <hash_params.h>
```

#### Data Fields

- `uint8_t` [schema](#) [4]  
*The schema identifier, on four bytes.*
- `uint8_t` [version](#) [2]  
*The version number, on two bytes.*
- `uint8_t` [reserved](#) [2]  
*Reserved, should be left zero according to the Skein specification.*
- `uint64_t` [out\\_len](#)
- `uint8_t` [unused](#) [16]  
*Unused, should be left zero according to the Skein specification.*

### 5.6.1 Detailed Description

Skein-256 hash function parameters.

#### Remarks

Refer to the Skein specification to know more about what each of these parameter fields stand for.

### 5.6.2 Field Documentation

#### 5.6.2.1 `uint64_t out_len`

Desired output length, in **bits**.

#### Remarks

This parameter affects the hash function's digest length.

The actual output length will be in bytes, and this parameter **will** be truncated to a byte boundary, so this should be a multiple of 8 to avoid any surprises.

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

- [/home/tom/Projects/github/Ordo/include/ordo/primitives/hash\\_functions/hash\\_params.h](#)



## 5.7 THREEFISH256\_PARAMS Struct Reference

Threefish-256 block cipher parameters.

```
#include <block_params.h>
```

### Data Fields

- uint64\_t [tweak](#) [2]

*The tweak word, on a pair of 64-bit words.*

### 5.7.1 Detailed Description

Threefish-256 block cipher parameters.

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

- /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_ciphers/[block\\_params.h](#)



## Chapter 6

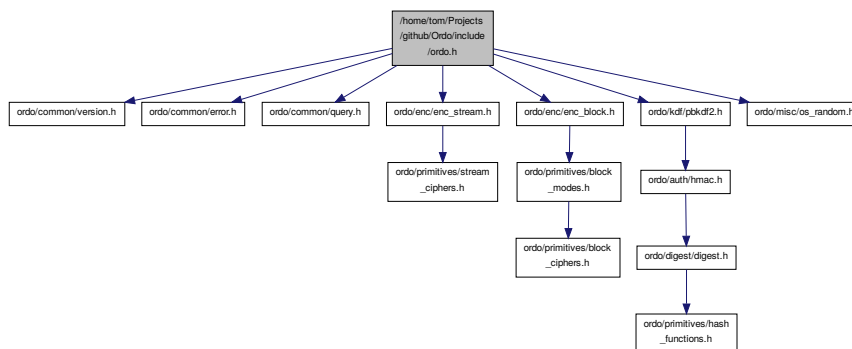
# File Documentation

### 6.1 /home/tom/Projects/github/Ordo/include/ordo.h File Reference

Wrapper.

```
#include "ordo/common/version.h"
#include "ordo/common/error.h"
#include "ordo/common/query.h"
#include "ordo/enc/enc_stream.h"
#include "ordo/enc/enc_block.h"
#include "ordo/kdf/pbkdf2.h"
#include "ordo/misc/os_random.h"
```

Include dependency graph for ordo.h:



### Functions

- ORDO\_PUBLIC void [ordo\\_allocator](#) (void \*(\*alloc)(size\_t, void \*), void(\*free)(void \*, void \*), void \*data)
- ORDO\_PUBLIC int [ordo\\_enc\\_block](#) (const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_params, const struct BLOCK\_MODE \*mode, const void \*mode\_params, int direction, const void \*key, size\_t key\_len, const void \*iv, size\_t iv\_len, const void \*in, size\_t in\_len, void \*out, size\_t \*out\_len)
- ORDO\_PUBLIC int [ordo\\_enc\\_stream](#) (const struct STREAM\_CIPHER \*cipher, const void \*params, const void \*key, size\_t key\_len, void \*inout, size\_t len)
- ORDO\_PUBLIC int [ordo\\_digest](#) (const struct HASH\_FUNCTION \*hash, const void \*params, const void \*in, size\_t in\_len, void \*digest)
- ORDO\_PUBLIC int [ordo\\_hmac](#) (const struct HASH\_FUNCTION \*hash, const void \*params, const void \*key, size\_t key\_len, const void \*in, size\_t in\_len, void \*fingerprint)

### 6.1.1 Detailed Description

Wrapper. This is the highest-level API for Ordo, which forgoes the use of cryptographic contexts completely, resulting in more concise code at the cost of reduced flexibility - in other words, if you can afford to use them, you probably want to do so.

This header also contains the `ordo_allocator()` function, which is used for (optionally) changing the memory allocator used by the library.

Usage snippet (compare to snippet in `digest.h`):

```
const char x[] = "Hello, world!";
unsigned char out[32];
int err = ordo_digest(sha256(), 0, x, strlen(x), out);
if (err) printf("Error encountered!");
// out = 315f5bdb76d0...
```

Some specialized headers are *not* included by this header - these are the endianness header & all primitive headers (their parameters are included), if you need their functionality please include them explicitly.

### 6.1.2 Function Documentation

#### 6.1.2.1 ORDO\_PUBLIC void ordo\_allocator ( void (\*)(size\_t, void \*) alloc, void (\*)(void \*, void \*) free, void \* data )

Replaces the default library memory allocator with a custom one.

##### Parameters

in	<i>alloc</i>	The allocation function.
in	<i>free</i>	The deallocation function.
in	<i>data</i>	Custom data passed to the above.

##### Remarks

After this function returns, all memory allocations done by the library will go through these functions instead. Do **not** use this function when the library has memory allocated with the current allocator, for obvious reasons. As a result this function should only be used at the start of the program, or at a point where you know the library to not be allocating any memory, e.g. there are no active contexts. Please ensure your allocator returns memory suitably aligned for the library to use - a 32-byte alignment is ideal, but a 16-byte alignment should suffice for most architectures. Calling this function with both arguments equal to 0 restores the default memory allocator (immediately ready for use).

#### 6.1.2.2 ORDO\_PUBLIC int ordo\_enc\_block ( const struct BLOCK\_CIPHER \* cipher, const void \* cipher\_params, const struct BLOCK\_MODE \* mode, const void \* mode\_params, int direction, const void \* key, size\_t key\_len, const void \* iv, size\_t iv\_len, const void \* in, size\_t in\_len, void \* out, size\_t \* out\_len )

Encrypts or decrypts data using a block cipher with a mode of operation.

##### Parameters

in	<i>cipher</i>	The block cipher to use.
in	<i>cipher_params</i>	The block cipher parameters.
in	<i>mode</i>	The mode of operation to use.
in	<i>mode_params</i>	The mode of operation parameters.

in	<i>direction</i>	1 for encryption, 0 for decryption.
in	<i>key</i>	The cryptographic key to use.
in	<i>key_len</i>	The length in bytes of the key.
in	<i>iv</i>	The initialization vector.
in	<i>iv_len</i>	The length in bytes of the IV.
in	<i>in</i>	The input plaintext/ciphertext buffer.
in	<i>in_len</i>	The length of the input buffer.
out	<i>out</i>	The output ciphertext/plaintext buffer.
out	<i>out_len</i>	The length of the output buffer.

**Returns**

[ORDO\\_SUCCESS](#) on success, else an error code.

**Remarks**

The `out` buffer should be large enough to accomodate the entire ciphertext which may be larger than the plaintext if a mode where padding is enabled and used, see padding notes in [enc\\_block.h](#).

**6.1.2.3** `ORDO_PUBLIC int ordo_enc_stream ( const struct STREAM_CIPHER * cipher, const void * params, const void * key, size_t key_len, void * inout, size_t len )`

Encrypts or decrypts data using a stream cipher.

**Parameters**

in	<i>cipher</i>	The stream cipher to use.
in	<i>params</i>	The stream cipher parameters.
in, out	<i>inout</i>	The plaintext or ciphertext buffer.
in	<i>len</i>	The length, in bytes, of the buffer.
in	<i>key</i>	The cryptographic key to use.
in	<i>key_len</i>	The length, in bytes, of the key.

**Returns**

[ORDO\\_SUCCESS](#) on success, else an error code.

**Remarks**

Stream ciphers do not strictly speaking require an initialization vector - if such a feature is needed, it is recommended to use a key derivation function to derive an encryption key from a master key using a pseudorandomly generated nonce.

Encryption is always done in place. If you require out-of-place encryption, make a copy of the plaintext prior to encryption.

By design, encryption and decryption are equivalent for stream ciphers - an implication is that encrypting a message twice using the same key yields the original message.

**6.1.2.4** `ORDO_PUBLIC int ordo_digest ( const struct HASH_FUNCTION * hash, const void * params, const void * in, size_t in_len, void * digest )`

Calculates the digest of a buffer using any hash function.

**Parameters**

in	<i>hash</i>	The hash function to use.
in	<i>params</i>	The hash function parameters.
in	<i>in</i>	The input buffer to hash.
in	<i>in_len</i>	The length in bytes of the buffer.
out	<i>digest</i>	The output buffer for the digest.

**Returns**

[ORDO\\_SUCCESS](#) on success, else an error code.

**6.1.2.5** `ORDO_PUBLIC int ordo_hmac ( const struct HASH_FUNCTION * hash, const void * params, const void * key, size_t key_len, const void * in, size_t in_len, void * fingerprint )`

Calculates the HMAC fingerprint of a buffer using any hash function.

**Parameters**

in	<i>hash</i>	The hash function to use.
in	<i>params</i>	The hash function parameters.
in	<i>key</i>	The key to use for authentication.
in	<i>key_len</i>	The length in bytes of the key.
in	<i>in</i>	The input buffer to authenticate.
in	<i>in_len</i>	The length, in bytes, of the input buffer.
out	<i>fingerprint</i>	The output buffer for the fingerprint.

**Returns**

[ORDO\\_SUCCESS](#) on success, else an error code.

**Remarks**

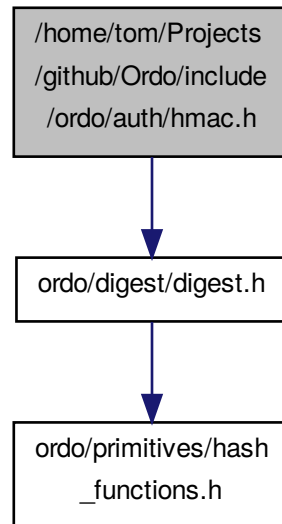
Do not use hash parameters which modify output length.

## 6.2 /home/tom/Projects/github/Ordo/include/ordo/auth/hmac.h File Reference

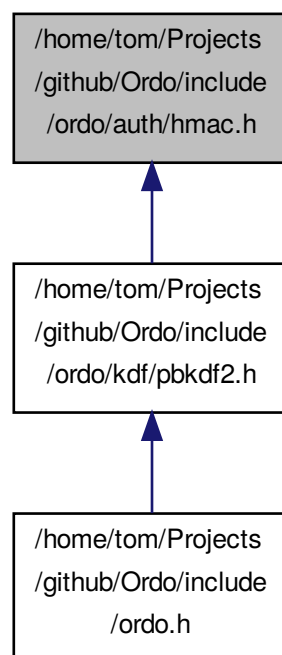
Module.

```
#include "ordo/digest/digest.h"
```

Include dependency graph for hmac.h:



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



## Functions

- ORDO\_PUBLIC struct HMAC\_CTX \* [hmac\\_alloc](#) (const struct HASH\_FUNCTION \*hash)
- ORDO\_PUBLIC int [hmac\\_init](#) (struct HMAC\_CTX \*ctx, const void \*key, size\_t key\_len, const void \*params)
- ORDO\_PUBLIC void [hmac\\_update](#) (struct HMAC\_CTX \*ctx, const void \*in, size\_t in\_len)
- ORDO\_PUBLIC int [hmac\\_final](#) (struct HMAC\_CTX \*ctx, void \*fingerprint)
- ORDO\_PUBLIC void [hmac\\_free](#) (struct HMAC\_CTX \*ctx)
- ORDO\_PUBLIC void [hmac\\_copy](#) (struct HMAC\_CTX \*dst, const struct HMAC\_CTX \*src)

### 6.2.1 Detailed Description

Module. Module for computing HMAC's (Hash-based Message Authentication Codes), which securely combine a hash function with a cryptographic key securely in order to provide both authentication and integrity, as per RFC 2104.

### 6.2.2 Function Documentation

#### 6.2.2.1 ORDO\_PUBLIC struct HMAC\_CTX\* [hmac\\_alloc](#) ( const struct HASH\_FUNCTION \* *hash* )

Allocates a new HMAC context.

##### Parameters

<i>in</i>	<i>hash</i>	The hash function to use.
-----------	-------------	---------------------------

##### Returns

The allocated HMAC context, or 0 if allocation fails.

##### Remarks

The PRF used for the HMAC will be the hash function as it behaves with default parameters. It is not possible to use hash function extensions (e.g. Skein in specialized HMAC mode) via this module, though if you intend to use a specific hash function you can just skip this abstraction layer and directly use whatever features it provides to compute message authentication codes.

#### 6.2.2.2 ORDO\_PUBLIC int [hmac\\_init](#) ( struct HMAC\_CTX \* *ctx*, const void \* *key*, size\_t *key\_len*, const void \* *params* )

Initializes an HMAC context, provided optional parameters.

##### Parameters

<i>in</i>	<i>ctx</i>	An allocated HMAC context.
<i>in</i>	<i>key</i>	The cryptographic key to use.
<i>in</i>	<i>key_len</i>	The size, in bytes, of the key.
<i>out</i>	<i>params</i>	Hash function specific parameters.

##### Returns

[ORDO\\_SUCCESS](#) on success, else an error code.

##### Remarks

The hash parameters apply to the inner hash operation only, which is the one used to hash the passed key with the inner mask.

Do not use hash parameters which modify the output length or this function's behavior is undefined.



6.2.2.3 ORDO\_PUBLIC void hmac\_update ( struct HMAC\_CTX \* *ctx*, const void \* *in*, size\_t *in\_len* )

Updates an HMAC context, feeding more data into it.

**Parameters**

in	<i>ctx</i>	An initialized HMAC context.
in	<i>in</i>	The data to feed into the context.
in	<i>in_len</i>	The length, in bytes, of the data.

**Remarks**

This function has the same properties, with respect to the input buffer, as the `digest_update()` function.

**6.2.2.4 ORDO\_PUBLIC int hmac\_final ( struct HMAC\_CTX \* ctx, void \* fingerprint )**

Finalizes a HMAC context, returning the final fingerprint.

**Parameters**

in	<i>ctx</i>	An initialized HMAC context.
out	<i>fingerprint</i>	The output buffer for the fingerprint.

**Returns**

`ORDO_SUCCESS` on success, else an error code.

**Remarks**

The fingerprint length is equal to the underlying hash function's digest length, which may be queried via `hash_digest_length()`.

**6.2.2.5 ORDO\_PUBLIC void hmac\_free ( struct HMAC\_CTX \* ctx )**

Frees a digest context.

**Parameters**

in	<i>ctx</i>	The HMAC context to be freed.
----	------------	-------------------------------

**Remarks**

The context need not have been initialized, but if it has been, it must have been finalized before calling this function.

Passing 0 to this function is valid, and will do nothing.

**6.2.2.6 ORDO\_PUBLIC void hmac\_copy ( struct HMAC\_CTX \* dst, const struct HMAC\_CTX \* src )**

Performs a deep copy of one context into another.

**Parameters**

out	<i>dst</i>	The destination context.
in	<i>src</i>	The source context.

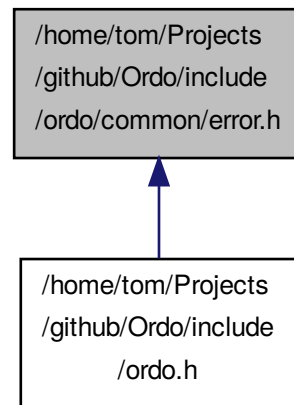
**Remarks**

The contexts must have been initialized using the exact same hash function with the exact same parameters, or this function invokes undefined behaviour.

## 6.3 /home/tom/Projects/github/Ordo/include/ordo/common/error.h File Reference

Utility.

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



### Enumerations

- enum `ORDO_ERROR` {  
`ORDO_SUCCESS`, `ORDO_FAIL`, `ORDO_LEFTOVER`, `ORDO_KEY_LEN`,  
`ORDO_PADDING`, `ORDO_ALLOC`, `ORDO_ARG` }

### Functions

- `ORDO_PUBLIC` const char \* `ordo_error_msg` (int code)

#### 6.3.1 Detailed Description

Utility. This header exposes error codes emitted by the library. Code which uses the library should always use the explicit error codes to check for errors, with the sole exception of `ORDO_SUCCESS` which is guaranteed to be zero.

#### 6.3.2 Enumeration Type Documentation

##### 6.3.2.1 enum `ORDO_ERROR`

Error codes used by the library.

##### Enumerator

**`ORDO_SUCCESS`** The function succeeded.

@remarks This is always defined as zero and is returned if a function encountered no error, unless specified otherwise.

**`ORDO_FAIL`** The function failed due to an external error.

@remarks This often indicates failure of an external component, such as the pseudorandom number generator provided by the OS (see #os\_random). The library is not responsible for this error.

**ORDO\_LEFTOVER** User input was left over unprocessed.

@remarks This applies to block cipher modes of operation for which padding has been disabled. If the input plaintext length is not a multiple of the cipher's block size, then the remaining incomplete block cannot be handled without padding, which is an error as it generally leads to inconsistent behavior on the part of the user.

**ORDO\_KEY\_LEN** The key length provided is invalid.

@remarks This occurs if you provide a key of an invalid length, such as passing a 128-bit key into a cipher which expects a 192-bit key. Primitives either have a range of possible key lengths (often characterized by a minimum and maximum key length, but this varies among algorithms) or only one specific key length. If you need to accept arbitrary length keys, you should consider hashing your key in some fashion before using it for encryption, for instance using a KDF.

@remarks The \c block\_cipher\_query() function can be used to select a suitable key length for a given block cipher via the \c #KEY\_LEN query code. For stream ciphers, use \c stream\_cipher\_query().

**ORDO\_PADDING** The padding was not recognized and decryption could not be completed.

@remarks This applies to block cipher modes for which padding is enabled. If the last block containing padding information is malformed, the padding will generally be unreadable and the correct message length cannot be retrieved, making correct decryption impossible. Note this is not guaranteed to occur if the padding block is corrupted. In other words, if \c #ORDO\_PADDING is returned, the padding block is certainly corrupted, however it may still be even if the library returns success (the returned plaintext will then be incorrect). If you \b must ensure the plaintext is decrypted correctly - and you probably should - you will want to use a MAC (Message Authentication Code) along with encryption, or an authenticated block cipher mode of operation.

**ORDO\_ALLOC** An attempt to allocate memory failed.

@remarks This occurs when the library's memory subsystem fails to allocate memory, and shouldn't occur during normal operation.

@remarks This likely indicates a memory leak in your code, though it may also be symptomatic of an error in the library's allocator (the default allocator uses malloc/free, but this can be overridden) or your own, if you changed the library's allocator at runtime.

**ORDO\_ARG** An invalid argument was passed to a function.

@remarks This is a generic error which is returned when the library finds an invalid parameter which would lead to inconsistent, undefined, or profoundly insecure behavior. Make sure your arguments are correct and do not contradict one another.

@remarks Keep in mind that the library cannot possibly catch all such errors, and you should still read the documentation if you are not sure what you are doing is valid.

## 6.3.3 Function Documentation

### 6.3.3.1 ORDO\_PUBLIC const char\* ordo\_error\_msg ( int code )

Generates a readable error message from an error code.

## Parameters

<code>in</code>	<code>code</code>	The error code to interpret.
-----------------	-------------------	------------------------------

## Returns

A null-terminated string containing the error description.

## Remarks

This function is intended for debugging purposes.

## 6.4 /home/tom/Projects/github/Ordo/include/ordo/common/interface.h File Reference

API.

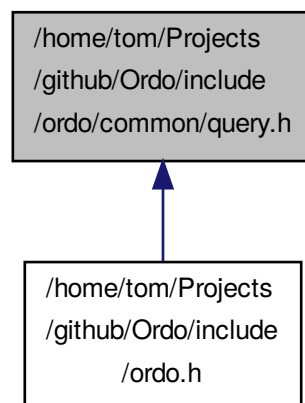
### 6.4.1 Detailed Description

API. This header contains some preprocessor definitions which try to abstract compiler-specific features (such as packing, export mechanisms, hot code sections), and will be included in every other header in the library.

## 6.5 /home/tom/Projects/github/Ordo/include/ordo/common/query.h File Reference

Utility.

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



## Enumerations

- enum `ORDO_QUERY` { `KEY_LEN_Q`, `BLOCK_SIZE_Q`, `DIGEST_LEN_Q`, `IV_LEN_Q` }

### 6.5.1 Detailed Description

Utility. This header contains declarations for query codes used when querying information from primitives or other library objects. The query must return a length or something relating to size, which is why it is used for key lengths and related quantities.

The query codes provide a lightweight mechanism to select suitable parameters when using the library, and, alternatively, iterating over all possible parameters when necessary, while still retaining some level of abstraction in user code.

All query functions take the following arguments:

- query code (one of the codes defined here)
- suggested value (type `size_t`)

They have the following properties (where X stands for the relevant quantity of the concerned primitive, e.g. "valid key length for some block cipher"):

- `query(code, 0)` returns the **smallest** X.
- `query(code, (size_t)-1)` returns the **largest** X.
- if `query(code, n) == n` then n is an X.
- if n is less than the largest X, then `query(code, n) > n`.
- if `query(code, n + 1) == n` then n is the **largest** X. Otherwise `query(code, n + 1)` returns the next X (in increasing order).

The motivation for designing this interface in this fashion is to ensure no information loss occurs when user input is provided to the library. For instance, if the user provides a 160-bit key to AES, he will first query the block cipher key length using `KEY_LEN_Q`, suggesting a 160-bit key, and the AES cipher will correctly identify the ideal key length as 192 bits, and not 128 bits (which would lead to part of the key being unused). This allows software using the library to dynamically adjust to whatever cryptographic primitives are in use without compromising security.

### 6.5.2 Enumeration Type Documentation

#### 6.5.2.1 enum `ORDO_QUERY`

Query codes used by the library. These end in `_Q`.

Enumerator

**`KEY_LEN_Q`** Query code to retrieve a key length.

Applicable to:  
 - block ciphers  
 - stream ciphers

**`BLOCK_SIZE_Q`** Query code to retrieve a block size.

Applicable to:  
 - block ciphers  
 - hash functions

@remarks For hash functions, this is taken to be the input size of the message block to the compression function, or, more formally, the amount of data necessary to trigger a compression function iteration. This may not be meaningful for all hash functions.

**`DIGEST_LEN_Q`** Query code to retrieve the default digest length of a hash function.

@remarks The suggested value is ignored for this query code.

Applicable to:  
 - hash functions

**IV\_LEN\_Q** Query code to retrieve an initialization vector length.

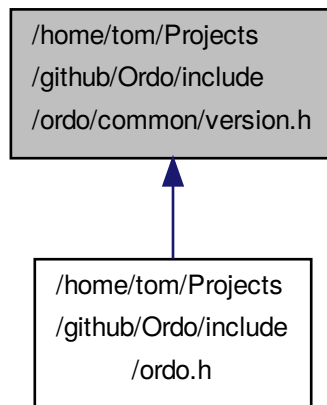
Applicable to:  
- block modes

@remarks As the block mode of operation primitives use block ciphers internally, the returned initialization vector length might depend on the block cipher (likely its block size).

## 6.6 /home/tom/Projects/github/Ordo/include/ordo/common/version.h File Reference

Utility.

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



### Data Structures

- struct `ORDO_VERSION`  
*Library version information.*

### Functions

- `ORDO_PUBLIC` const struct `ORDO_VERSION * ordo_version` (void)  
*Returns an `ORDO_VERSION` structure for this library.*

#### 6.6.1 Detailed Description

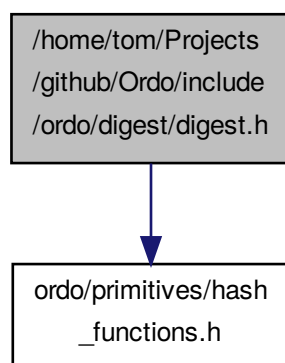
Utility. This header exposes functionality relating to the library's version.

## 6.7 /home/tom/Projects/github/Ordo/include/ordo/digest/digest.h File Reference

Module.

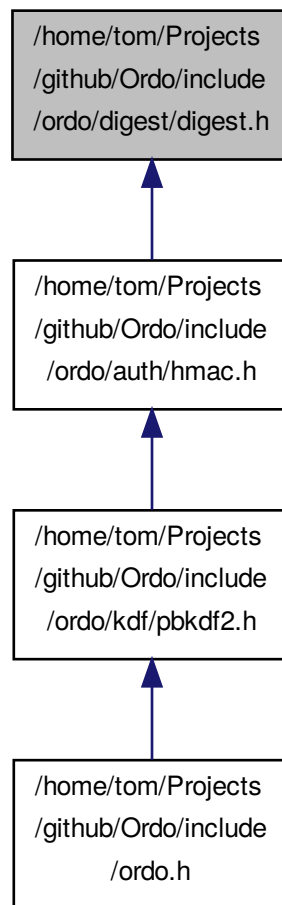
```
#include "ordo/primitives/hash_functions.h"
```

Include dependency graph for digest.h:





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



## Functions

- ORDO\_PUBLIC struct DIGEST\_CTX \* [digest\\_alloc](#) (const struct HASH\_FUNCTION \*hash)
- ORDO\_PUBLIC int [digest\\_init](#) (struct DIGEST\_CTX \*ctx, const void \*params)
- ORDO\_PUBLIC void [digest\\_update](#) (struct DIGEST\_CTX \*ctx, const void \*in, size\_t in\_len)
- ORDO\_PUBLIC void [digest\\_final](#) (struct DIGEST\_CTX \*ctx, void \*digest)
- ORDO\_PUBLIC void [digest\\_free](#) (struct DIGEST\_CTX \*ctx)
- ORDO\_PUBLIC void [digest\\_copy](#) (struct DIGEST\_CTX \*dst, const struct DIGEST\_CTX \*src)
- ORDO\_PUBLIC size\_t [digest\\_length](#) (const struct HASH\_FUNCTION \*hash)

### 6.7.1 Detailed Description

Module. Module to compute cryptographic digests, using cryptographic hash function primitives (as a pointer to a `HASH_FUNCTION` structure).

The advantage of using this digest module instead of the hash function abstraction layer is this keeps track of the hash function primitive for you within an opaque `DIGEST_CTX` context structure, simplifying code and making it less error-prone.

Usage snippet:

```
const struct HASH_FUNCTION *hash = sha256();
struct DIGEST_CTX *ctx = digest_alloc(hash);
if (!ctx) printf("Failed to allocate ctx!");

int err = digest_init(ctx, 0);
if (err) printf("Got error!");

const char x[] = "Hello, world!";
digest_update(ctx, x, strlen(x));

unsigned char out[32];
digest_final(ctx, out);
// out = 315f5bdb76d0...

digest_free(ctx);
```

## 6.7.2 Function Documentation

### 6.7.2.1 ORDO\_PUBLIC struct DIGEST\_CTX\* digest\_alloc ( const struct HASH\_FUNCTION \* *hash* )

Allocates a new DIGEST\_CTX (digest context).

Parameters

in	<i>hash</i>	The hash function primitive to use.
----	-------------	-------------------------------------

Returns

The allocated digest context, or 0 if allocation fails.

### 6.7.2.2 ORDO\_PUBLIC int digest\_init ( struct DIGEST\_CTX \* *ctx*, const void \* *params* )

Initializes a digest context.

Parameters

in, out	<i>ctx</i>	An allocated digest context.
in	<i>params</i>	Hash function parameters.

Returns

ORDO\_SUCCESS on success, else an error code.

Remarks

It is always valid to pass 0 into *params* if you don't want to use special features offered by a specific hash function.

It is **not** valid to initialize digest contexts more than once before calling `digest_final()`, this is because some algorithms may allocate additional memory depending on the parameters given.

### 6.7.2.3 ORDO\_PUBLIC void digest\_update ( struct DIGEST\_CTX \* *ctx*, const void \* *in*, size\_t *in\_len* )

Feeds data into a digest context.

**Parameters**

<i>in, out</i>	<i>ctx</i>	An initialized digest context.
<i>in</i>	<i>in</i>	The data to feed into the context.
<i>in</i>	<i>in_len</i>	The length, in bytes, of the data.

**Remarks**

This function has the same property as [hash\\_function\\_update\(\)](#), with respect to calling it in succession with different buffers.

It is valid to pass a zero-length buffer (`in_len == 0`), which will do nothing (if this is the case, `in` may be 0).

**6.7.2.4 ORDO\_PUBLIC void digest\_final ( struct DIGEST\_CTX \* ctx, void \* digest )**

Finalizes a digest context, returning the digest of all the data fed into it through successive [digest\\_update\(\)](#) calls.

**Parameters**

<i>in, out</i>	<i>ctx</i>	An initialized digest context.
<i>out</i>	<i>digest</i>	The output buffer for the digest.

**Remarks**

The `digest` buffer should be large enough to accomodate the digest - you can query the hash function's default digest length in bytes by the [digest\\_length\(\)](#) function, note if you provided parameters which modify the hash function's digest length, then you should already know how long the digest will be (refer to the parameter's documentation).

Calling this function immediately after [digest\\_init\(\)](#) is valid and will return the so-called "zero-length" digest, which is the digest of the input of length zero.

After this function returns, you may not call [digest\\_update\(\)](#) again until you reinitialize the context using [digest\\_init\(\)](#).

**6.7.2.5 ORDO\_PUBLIC void digest\_free ( struct DIGEST\_CTX \* ctx )**

Frees a digest context.

**Parameters**

<i>in</i>	<i>ctx</i>	The digest context to be freed.
-----------	------------	---------------------------------

**Remarks**

The context need not have been initialized, but if it has been, it must have been finalized before calling this function.

Passing 0 to this function is valid, and will do nothing.

**6.7.2.6 ORDO\_PUBLIC void digest\_copy ( struct DIGEST\_CTX \* dst, const struct DIGEST\_CTX \* src )**

Performs a deep copy of one context into another.

**Parameters**

out	<i>dst</i>	The destination context.
in	<i>src</i>	The source context.

**Remarks**

The contexts must have been initialized using the exact same hash function with the exact same parameters, or this function invokes undefined behaviour.

This function is useful when hashing many messages with a common prefix, where the state of the digest context after having been fed the prefix can be saved and then reused multiple times.

**6.7.2.7 ORDO\_PUBLIC size\_t digest\_length ( const struct HASH\_FUNCTION \* hash )**

Returns the default digest length of a hash function.

**Parameters**

in	<i>hash</i>	A hash function primitive.
----	-------------	----------------------------

**Returns**

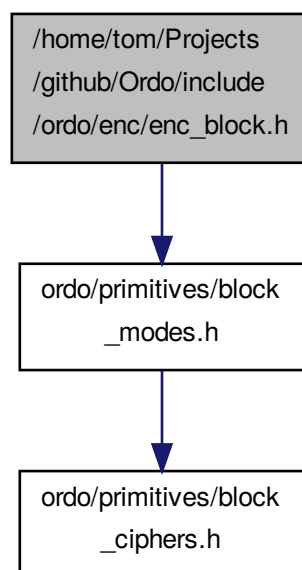
The length of the digest to be written in the `digest` parameter of `digest_final()`, if no parameters which affect output length were provided to `digest_init()`.

**6.8 /home/tom/Projects/github/Ordo/include/ordo/enc/enc\_block.h File Reference**

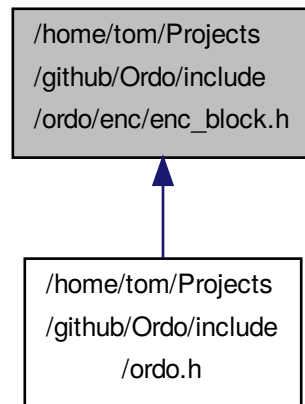
Module.

```
#include "ordo/primitives/block_modes.h"
```

Include dependency graph for `enc_block.h`:



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



## Functions

- ORDO\_PUBLIC struct ENC\_BLOCK\_CTX \* [enc\\_block\\_alloc](#) (const struct BLOCK\_CIPHER \*cipher, const struct BLOCK\_MODE \*mode)
- ORDO\_PUBLIC int [enc\\_block\\_init](#) (struct ENC\_BLOCK\_CTX \*ctx, const void \*key, size\_t key\_len, const void \*iv, size\_t iv\_len, int direction, const void \*cipher\_params, const void \*mode\_params)
- ORDO\_PUBLIC void [enc\\_block\\_update](#) (struct ENC\_BLOCK\_CTX \*ctx, const void \*in, size\_t in\_len, void \*out, size\_t \*out\_len)
- ORDO\_PUBLIC int [enc\\_block\\_final](#) (struct ENC\_BLOCK\_CTX \*ctx, void \*out, size\_t \*out\_len)
- ORDO\_PUBLIC void [enc\\_block\\_free](#) (struct ENC\_BLOCK\_CTX \*ctx)
- ORDO\_PUBLIC void [enc\\_block\\_copy](#) (struct ENC\_BLOCK\_CTX \*dst, const struct ENC\_BLOCK\_CTX \*src)
- ORDO\_PUBLIC size\_t [enc\\_block\\_key\\_len](#) (const struct BLOCK\_CIPHER \*cipher, size\_t key\_len)
- ORDO\_PUBLIC size\_t [enc\\_block\\_iv\\_len](#) (const struct BLOCK\_CIPHER \*cipher, const struct BLOCK\_MODE \*mode, size\_t iv\_len)

### 6.8.1 Detailed Description

Module. Module to encrypt plaintext and decrypt ciphertext with different block ciphers and modes of operation. Note it is always possible to skip this API and directly use the lower-level functions available in the individual mode of operation headers, but this interface abstracts away some of the more boilerplate details and so should be preferred.

If you wish to use the lower level API, you will need to manage your block cipher contexts yourself, which can give more flexibility in some particular cases but is often unnecessary.

The padding algorithm for modes of operation which use padding is PKCS7 (RFC 5652), which appends N bytes of value N, where N is the number of padding bytes required, in bytes (between 1 and the block cipher's block size).

### 6.8.2 Function Documentation

6.8.2.1 ORDO\_PUBLIC struct ENC\_BLOCK\_CTX\* enc\_block\_alloc ( const struct BLOCK\_CIPHER \* *cipher*, const struct BLOCK\_MODE \* *mode* )

Allocates a new block encryption context.

## Parameters

in	<i>cipher</i>	The block cipher to use.
in	<i>mode</i>	The block mode of operation to use.

## Returns

The allocated block encryption context, or 0 if an allocation error occurred.

**6.8.2.2** `ORDO_PUBLIC int enc_block_init ( struct ENC_BLOCK_CTX * ctx, const void * key, size_t key_len, const void * iv, size_t iv_len, int direction, const void * cipher_params, const void * mode_params )`

Initializes a block encryption context.

## Parameters

in, out	<i>ctx</i>	A block encryption context.
in	<i>key</i>	The cryptographic key to use.
in	<i>key_len</i>	The length, in bytes, of the key.
in	<i>iv</i>	The initialization vector to use.
in	<i>iv_len</i>	The length, in bytes, of the IV.
in	<i>direction</i>	1 for encryption, 0 for decryption.
in	<i>cipher_params</i>	Block cipher specific parameters.
in	<i>mode_params</i>	Mode of operation specific parameters.

## Returns

[ORDO\\_SUCCESS](#) on success, else an error code.

## Remarks

The initialization vector may be 0, if the mode of operation does not require one - consult the documentation of the mode to know what it expects.

**6.8.2.3** `ORDO_PUBLIC void enc_block_update ( struct ENC_BLOCK_CTX * ctx, const void * in, size_t in_len, void * out, size_t * out_len )`

Encrypts or decrypts a data buffer.

## Parameters

in, out	<i>ctx</i>	A block encryption context.
in	<i>in</i>	The plaintext or ciphertext buffer.
in	<i>in_len</i>	Length, in bytes, of the input buffer.
out	<i>out</i>	The ciphertext or plaintext buffer.
out	<i>out_len</i>	The number of bytes written to <i>out</i> .

## Remarks

This function might not immediately encrypt all data fed into it, and will write the amount of input bytes effectively encrypted in *out\_len*. However, it does **not** mean that the plaintext left over has been "rejected" or "ignored". It **has** been taken into account but the corresponding ciphertext simply can't be produced until more data is fed into it (or until [enc\\_block\\_final\(\)](#) is called).

Some modes of operation always process all input data, in which case they may allow *out\_len* to be nil; check the documentation

6.8.2.4 ORDO\_PUBLIC int enc\_block\_final ( struct ENC\_BLOCK\_CTX \* *ctx*, void \* *out*, size\_t \* *out\_len* )

Finalizes a block encryption context.



**Parameters**

<i>in, out</i>	<i>ctx</i>	A block encryption context.
<i>out</i>	<i>out</i>	The ciphertext or plaintext buffer.
<i>out</i>	<i>out_len</i>	The number of bytes written to <i>out</i> .

**Returns**

`ORDO_SUCCESS` on success, else an error code.

**Remarks**

The function will return up to one block size's worth of data and may not return any data at all. For example, for the CBC mode of operation (with padding on), this function will, for encryption, append padding bytes to the final plaintext block, and return the padding block, whereas for decryption, it will take that padding block and strip the padding off, returning the last few bytes of plaintext.

Some modes of operation always process all input data, in which case they may allow *out\_len* to be nil; check the documentation

**6.8.2.5 ORDO\_PUBLIC void enc\_block\_free ( struct ENC\_BLOCK\_CTX \* ctx )**

Frees a block encryption context.

**Parameters**

<i>in, out</i>	<i>ctx</i>	A block encryption context.
----------------	------------	-----------------------------

**6.8.2.6 ORDO\_PUBLIC void enc\_block\_copy ( struct ENC\_BLOCK\_CTX \* dst, const struct ENC\_BLOCK\_CTX \* src )**

Performs a deep copy of one context into another.

**Parameters**

<i>out</i>	<i>dst</i>	The destination context.
<i>in</i>	<i>src</i>	The source context.

**Remarks**

The contexts must have been allocated with the same block cipher, block mode, and the exact same parameters for both - unless the parameter documentation states otherwise - else this function's behavior is undefined.

**6.8.2.7 ORDO\_PUBLIC size\_t enc\_block\_key\_len ( const struct BLOCK\_CIPHER \* cipher, size\_t key\_len )**

Queries the key length of a block cipher.

**Parameters**

<i>in</i>	<i>cipher</i>	A block cipher primitive.
<i>in</i>	<i>key_len</i>	A suggested key length.

**Returns**

An ideal key length to use for this cipher.

6.8.2.8 ORDO\_PUBLIC size\_t enc\_block\_iv\_len ( const struct BLOCK\_CIPHER \* *cipher*, const struct BLOCK\_MODE \* *mode*, size\_t *iv\_len* )

Queries the IV length of a block mode and block cipher.

## Parameters

in	<i>cipher</i>	A block cipher primitive.
in	<i>mode</i>	A block mode primitive.
in	<i>iv_len</i>	A suggested IV length.

## Returns

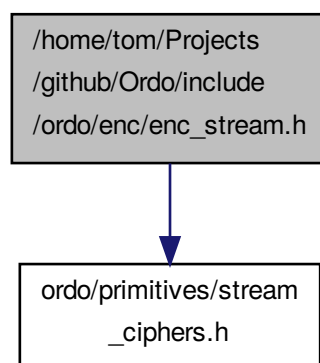
An ideal IV length to use for this mode and cipher.

## 6.9 /home/tom/Projects/github/Ordo/include/ordo/enc/enc\_stream.h File Reference

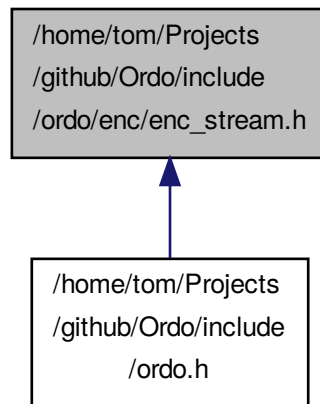
Module.

```
#include "ordo/primitives/stream_ciphers.h"
```

Include dependency graph for enc\_stream.h:



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



## Functions

- ORDO\_PUBLIC struct ENC\_STREAM\_CTX \* [enc\\_stream\\_alloc](#) (const struct STREAM\_CIPHER \*cipher)
- ORDO\_PUBLIC int [enc\\_stream\\_init](#) (struct ENC\_STREAM\_CTX \*ctx, const void \*key, size\_t key\_size, const void \*params)
- ORDO\_PUBLIC void [enc\\_stream\\_update](#) (struct ENC\_STREAM\_CTX \*ctx, void \*buffer, size\_t len)
- ORDO\_PUBLIC void [enc\\_stream\\_final](#) (struct ENC\_STREAM\_CTX \*ctx)
- ORDO\_PUBLIC void [enc\\_stream\\_free](#) (struct ENC\_STREAM\_CTX \*ctx)
- ORDO\_PUBLIC void [enc\\_stream\\_copy](#) (struct ENC\_STREAM\_CTX \*dst, const struct ENC\_STREAM\_CTX \*src)
- ORDO\_PUBLIC size\_t [enc\\_stream\\_key\\_len](#) (const struct STREAM\_CIPHER \*cipher, size\_t key\_len)

### 6.9.1 Detailed Description

Module. Interface to encrypt plaintext and decrypt ciphertext with various stream ciphers.

### 6.9.2 Function Documentation

#### 6.9.2.1 ORDO\_PUBLIC struct ENC\_STREAM\_CTX\* [enc\\_stream\\_alloc](#) ( const struct STREAM\_CIPHER \* *cipher* )

Allocates a new stream encryption context.

Parameters

<i>in</i>	<i>cipher</i>	The stream cipher to use.
-----------	---------------	---------------------------

Returns

The allocated stream encryption context, or 0 if an allocation error occurred.

6.9.2.2 ORDO\_PUBLIC int enc\_stream\_init ( struct ENC\_STREAM\_CTX \* *ctx*, const void \* *key*, size\_t *key\_size*, const void \* *params* )

Initializes a stream encryption context.

**Parameters**

in, out	<i>ctx</i>	A stream encryption context.
in	<i>key</i>	The cryptographic key to use.
in	<i>key_size</i>	The size, in bytes, of the key.
in	<i>params</i>	Stream cipher specific parameters.

**Returns**

`ORDO_SUCCESS` on success, else an error code.

**6.9.2.3 ORDO\_PUBLIC void enc\_stream\_update ( struct ENC\_STREAM\_CTX \* ctx, void \* buffer, size\_t len )**

Encrypts or decrypts a data buffer.

**Parameters**

in, out	<i>ctx</i>	A stream encryption context.
in, out	<i>buffer</i>	The plaintext or ciphertext buffer.
in	<i>len</i>	Number of bytes to read from the buffer.

**Remarks**

By nature, stream ciphers encrypt and decrypt data the same way, in other words, if you encrypt data twice, you will get back the original data.

Stream encryption is always done in place by design.

**6.9.2.4 ORDO\_PUBLIC void enc\_stream\_final ( struct ENC\_STREAM\_CTX \* ctx )**

Finalizes a stream encryption context.

**Parameters**

in, out	<i>ctx</i>	A stream encryption context.
---------	------------	------------------------------

**6.9.2.5 ORDO\_PUBLIC void enc\_stream\_free ( struct ENC\_STREAM\_CTX \* ctx )**

Frees a stream encryption context.

**Parameters**

in, out	<i>ctx</i>	A stream encryption context.
---------	------------	------------------------------

**6.9.2.6 ORDO\_PUBLIC void enc\_stream\_copy ( struct ENC\_STREAM\_CTX \* dst, const struct ENC\_STREAM\_CTX \* src )**

Performs a deep copy of one context into another.

**Parameters**

out	<i>dst</i>	The destination context.
in	<i>src</i>	The source context.

**Remarks**

Both the contexts must have been allocated with the same stream cipher, and the exact same parameters - unless the parameter documentation states otherwise - else this function's behavior is undefined.

6.9.2.7 ORDO\_PUBLIC size\_t enc\_stream\_key\_len ( const struct STREAM\_CIPHER \* *cipher*, size\_t *key\_len* )

Queries a stream cipher for its key length.

## Parameters

in	<i>cipher</i>	The stream cipher to probe.
in	<i>key_len</i>	A suggested key length.

## Returns

*key\_len* if and only if *key\_len* is a valid key length for this stream cipher. Otherwise, returns the nearest valid key length greater than *key\_len*. However, if no such key length exists, it will return the largest key length admitted by the stream cipher.

## 6.10 /home/tom/Projects/github/Ordo/include/ordo/internal/alg.h File Reference

### Internal, Utility

#### Macros

- #define [bits](#)(n)
- #define [bytes](#)(n)
- #define [offset](#)(ptr, len)

#### Functions

- ORDO\_HIDDEN int [pad\\_check](#) (const unsigned char \*buffer, uint8\_t padding)
- ORDO\_HIDDEN void [xor\\_buffer](#) (void \*dst, const void \*src, size\_t len)
- ORDO\_HIDDEN void [inc\\_buffer](#) (unsigned char \*buffer, size\_t len)

### 6.10.1 Detailed Description

**Internal, Utility** This header provides various utility functions which are used by some library modules and a few convenience macros. It is not to be used outside the library, and this is enforced by an include guard. If you really must access it, define the `ORDO_INTERNAL_ACCESS` token before including it.

Functions in internal headers are not prefixed, be wary of name clashes.

### 6.10.2 Macro Definition Documentation

#### 6.10.2.1 #define [bits](#)( *n* )

Converts bits into bytes (rounded down to the nearest byte boundary).

#### Remarks

As an example, [bits](#) (256) returns 32 (bytes).

#### 6.10.2.2 #define [bytes](#)( *n* )

Converts bytes into bits (as a multiple of 8 bits).

#### Remarks

As an example, [bytes](#) (32) returns 256 (bits).



### 6.10.2.3 `#define offset( ptr, len )`

Computes a byte-based offset.

**Parameters**

in	<i>ptr</i>	Base pointer.
in	<i>len</i>	Offset (in bytes).

**Returns**

The pointer exactly `len` bytes after `ptr`.

**Remarks**

This is a dangerous macro, in the sense it can lead to accessing data at unaligned addresses, and so should be used carefully.

**6.10.3 Function Documentation****6.10.3.1 ORDO\_HIDDEN int pad\_check ( const unsigned char \* *buffer*, uint8\_t *padding* )**

Checks whether a buffer conforms to PKCS padding.

**Parameters**

in	<i>buffer</i>	The buffer to check, starting at the first padding byte.
in	<i>padding</i>	The padding byte value to check this buffer against (between 1 and 255).

**Returns**

1 if the buffer is valid, 0 otherwise.

**Remarks**

PKCS padding is defined as appending `N` bytes of padding data at the end of the message, each with binary value `N`, with `N` between 1 and the block size of the block cipher used such that the length of the message plus `N` is a multiple of the block cipher's block size.

This implies the buffer must be at least `padding` bytes long.

**6.10.3.2 ORDO\_HIDDEN void xor\_buffer ( void \* *dst*, const void \* *src*, size\_t *len* )**

Performs a bitwise exclusive-or of one buffer onto another.

**Parameters**

in, out	<i>dst</i>	The destination buffer.
in	<i>src</i>	The source buffer.
in	<i>len</i>	The number of bytes to process.

**Remarks**

This is conceptually equivalent to `dst ^ src`.

The Source and destination buffers may be the same (in which case the buffer will contain `len` zeroes), but otherwise they cannot overlap.

**6.10.3.3 ORDO\_HIDDEN void inc\_buffer ( unsigned char \* *buffer*, size\_t *len* )**

Increments a buffer of arbitrary length, as though it were a `len` byte integer stored as a byte array.

## Parameters

<i>in, out</i>	<i>buffer</i>	The buffer to increment in-place.
<i>in</i>	<i>len</i>	The size, in bytes, of the buffer.

## Remarks

Carry propagation is done left-to-right.

## 6.11 /home/tom/Projects/github/Ordo/include/ordo/internal/implementation.h File Reference

**Internal, API**

### 6.11.1 Detailed Description

**Internal, API** This header contains some compiler-dependent macros, for defining various semantics which the users of this library should not depend on. It is an error to include this header in any code outside the Ordo implementation.

Every source file will include this header.

## 6.12 /home/tom/Projects/github/Ordo/include/ordo/internal/mem.h File Reference

**Internal, Utility**

## Functions

- ORDO\_HIDDEN void \* [mem\\_alloc](#) (size\_t size)
- ORDO\_HIDDEN void [mem\\_free](#) (void \*ptr)
- ORDO\_HIDDEN void [mem\\_erase](#) (void \*ptr, size\_t size)

### 6.12.1 Detailed Description

**Internal, Utility** Contains the library's memory manager. The library relies solely on this on this interface to allocate cryptographic contexts. This header should not be used outside the library, this is enforced by an include guard.

If you are just trying to change the allocator used, this is now provided elsewhere, in the [ordo.h](#) header - see [ordo\\_allocator\(\)](#).

See [alg.h](#) about internal headers.

### 6.12.2 Function Documentation

#### 6.12.2.1 ORDO\_HIDDEN void\* mem\_alloc ( size\_t size )

Allocates a memory buffer.

## Parameters

<i>in</i>	<i>size</i>	The amount of memory required, in bytes.
-----------	-------------	--

**Returns**

A pointer to the allocated memory on success, or 0 if the function fails to allocate the requested amount of memory.

**Remarks**

Memory may be left uninitialized upon allocation.  
 Memory returned by the function is expected to be aligned for all possible uses by the library.  
 This function is thread-safe.

**6.12.2.2 ORDO\_HIDDEN void mem\_free ( void \* *ptr* )**

Deallocates a memory buffer.

**Parameters**

<i>in</i>	<i>ptr</i>	A memory buffer to free.
-----------	------------	--------------------------

**Remarks**

Passing 0 to this function is valid and will do nothing.  
 The memory buffer must have been allocated with `mem_alloc()`.  
 This function is thread-safe.

**6.12.2.3 ORDO\_HIDDEN void mem\_erase ( void \* *ptr*, size\_t *size* )**

Overwrites a memory buffer with zeroes.

**Parameters**

<i>in, out</i>	<i>ptr</i>	The memory buffer to overwrite.
<i>in</i>	<i>size</i>	The number of bytes to overwrite.

**6.13 /home/tom/Projects/github/Ordo/include/ordo/internal/sys.h File Reference**

**Internal**, Utility

**6.13.1 Detailed Description**

**Internal**, Utility This header provides system-dependent functionality and is internal to the library. It probably shouldn't ever be used from outside the library.

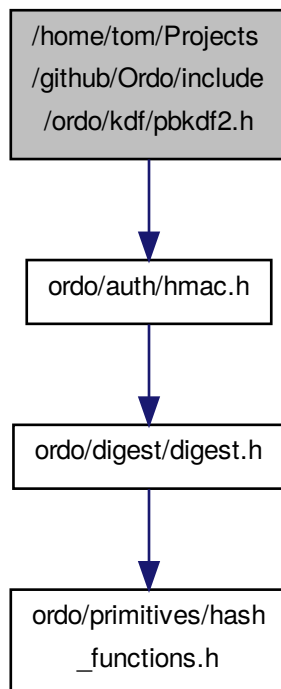
See [alg.h](#) about internal headers.

**6.14 /home/tom/Projects/github/Ordo/include/ordo/kdf/pbkdf2.h File Reference**

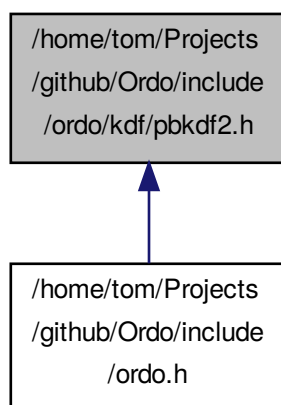
Module.

```
#include "ordo/auth/hmac.h"
```

Include dependency graph for pbkdf2.h:



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



## Functions

- `ORDO_PUBLIC int pbkdf2 (const struct HASH_FUNCTION *hash, const void *params, const void *password, size_t password_len, const void *salt, size_t salt_len, size_t iterations, void *out, size_t out_len)`

### 6.14.1 Detailed Description

Module. Module for the PBKDF2 algorithm (Password-Based Key Derivation Function v2) which combines a keyed PRF (here HMAC) with a salt in order to generate secure cryptographic keys, as per RFC 2898. Also features a variable iteration count (work factor) to help thwart brute-force attacks.

Unlike most other cryptographic modules, the PBKDF2 API does not follow the traditional init/update/final pattern but is a context-free function as its inputs are almost always known in advance. As such this module does not benefit from the use of contexts.

### 6.14.2 Function Documentation

6.14.2.1 `ORDO_PUBLIC int pbkdf2 ( const struct HASH_FUNCTION * hash, const void * params, const void * password, size_t password_len, const void * salt, size_t salt_len, size_t iterations, void * out, size_t out_len )`

Derives a key using PBKDF2.

#### Parameters

in	<i>hash</i>	The hash function to use (the PRF used will be an instantiation of HMAC with it)
in	<i>params</i>	Hash-specific parameters.
in	<i>password</i>	The password to derive a key from.
in	<i>password_len</i>	The length in bytes of the password.
in	<i>salt</i>	The cryptographic salt to use.
in	<i>salt_len</i>	The length in bytes of the salt.
in	<i>iterations</i>	The number of PBKDF2 iterations to use.
out	<i>out</i>	The output buffer for the derived key.
in	<i>out_len</i>	The required length, in bytes, of the key.

#### Returns

`ORDO_SUCCESS` on success, else an error code.

#### Remarks

There is a maximum output length of  $2^{32} - 1$  multiplied by the digest length of the chosen hash function, but it is unlikely to be reached as derived keys are generally no longer than a few hundred bits. Reaching the limit will result in an `ORDO_ARG` error code. This limit is mandated by the PBKDF2 specification.

The `out` buffer should be at least `out_len` bytes long.

Do not use hash parameters which modify the output length or this function's behavior is undefined.

## 6.15 /home/tom/Projects/github/Ordo/include/ordo/misc/endianness.h File Reference

Utility.

### 6.15.1 Detailed Description

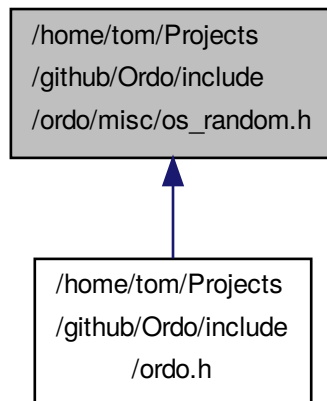
Utility. This header provides endianness functionality. You may use it freely as it has a stable API and is public. Only supports little/big endian for now.

The functions in this header are not prefixed, be wary of name clashes.

## 6.16 /home/tom/Projects/github/Ordo/include/ordo/misc/os\_random.h File Reference

Module.

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



### Functions

- ORDO\_PUBLIC int `os_random` (void \*out, size\_t len)

#### 6.16.1 Detailed Description

Module. Exposes the OS CSPRNG (Cryptographically Secure PseudoRandom Number Generator) interface, which is basically a cross-platform wrapper to the OS-provided entropy pool.

- **Linux:** Reads from `/dev/urandom`.
- **Windows:** Acquires a CSP token and calls `CryptGenRandom`.

#### 6.16.2 Function Documentation

##### 6.16.2.1 ORDO\_PUBLIC int os\_random ( void \* out, size\_t len )

Generates cryptographically secure pseudorandom numbers.

Parameters

out	out	The destination buffer.
-----	-----	-------------------------

<code>in</code>	<code>len</code>	The number of bytes to generate.
-----------------	------------------	----------------------------------

### Returns

`ORDO_SUCCESS` on success, else an error code.

### Remarks

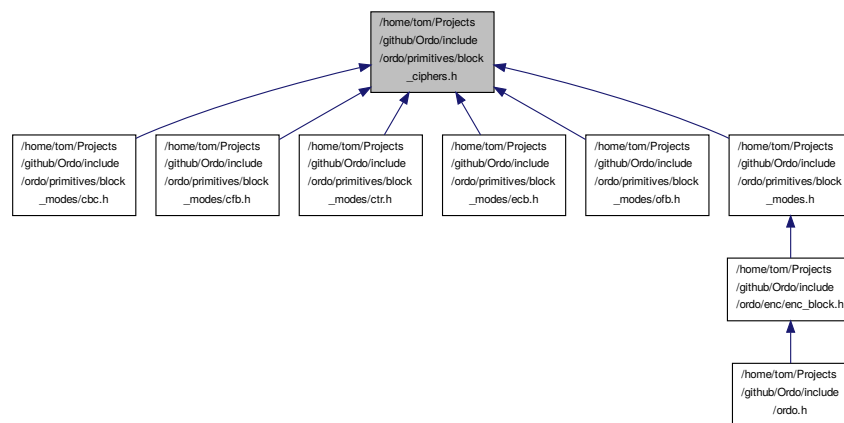
This function uses the CSPRNG provided by your operating system.

If the platform does not provide this feature, this function will always fail with the `ORDO_FAIL` error message, and any data in the buffer should be discarded as indeterminate.

## 6.17 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_ciphers.h File Reference

Abstraction Layer.

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



### Functions

- `ORDO_PUBLIC` const char \* `block_cipher_name` (const struct BLOCK\_CIPHER \*primitive)
- `ORDO_PUBLIC` const struct BLOCK\_CIPHER \* `nullcipher` (void)  
The NullCipher block cipher.
- `ORDO_PUBLIC` const struct BLOCK\_CIPHER \* `threefish256` (void)  
The Threefish-256 block cipher.
- `ORDO_PUBLIC` const struct BLOCK\_CIPHER \* `aes` (void)  
The AES block cipher.
- `ORDO_PUBLIC` size\_t `block_cipher_count` (void)
- `ORDO_PUBLIC` const struct BLOCK\_CIPHER \* `block_cipher_by_name` (const char \*name)
- `ORDO_PUBLIC` const struct BLOCK\_CIPHER \* `block_cipher_by_index` (size\_t index)



- ORDO\_PUBLIC void \* [block\\_cipher\\_alloc](#) (const struct BLOCK\_CIPHER \*primitive)
- ORDO\_PUBLIC int [block\\_cipher\\_init](#) (const struct BLOCK\_CIPHER \*primitive, void \*state, const void \*key, size\_t key\_len, const void \*params)
- ORDO\_PUBLIC void [block\\_cipher\\_forward](#) (const struct BLOCK\_CIPHER \*primitive, const void \*state, void \*block)
- ORDO\_PUBLIC void [block\\_cipher\\_inverse](#) (const struct BLOCK\_CIPHER \*primitive, const void \*state, void \*block)
- ORDO\_PUBLIC void [block\\_cipher\\_final](#) (const struct BLOCK\_CIPHER \*primitive, void \*state)
- ORDO\_PUBLIC void [block\\_cipher\\_free](#) (const struct BLOCK\_CIPHER \*primitive, void \*state)
- ORDO\_PUBLIC void [block\\_cipher\\_copy](#) (const struct BLOCK\_CIPHER \*primitive, void \*dst, const void \*src)
- ORDO\_PUBLIC size\_t [block\\_cipher\\_query](#) (const struct BLOCK\_CIPHER \*primitive, int query, size\_t value)

### 6.17.1 Detailed Description

Abstraction Layer. This abstraction layer declares all the block ciphers, and also makes them available to higher level modules. This does not actually do encryption at all but simply abstracts block cipher permutations, the encryption modules are in the `enc` folder: [enc\\_block.h](#).

### 6.17.2 Function Documentation

#### 6.17.2.1 ORDO\_PUBLIC const char\* block\_cipher\_name ( const struct BLOCK\_CIPHER \* *primitive* )

Returns the name of a block cipher primitive.

##### Parameters

<i>in</i>	<i>primitive</i>	A block cipher primitive.
-----------	------------------	---------------------------

##### Returns

Returns the block cipher's name.

##### Remarks

This name can then be used in [block\\_cipher\\_by\\_name\(\)](#).

#### 6.17.2.2 ORDO\_PUBLIC size\_t block\_cipher\_count ( void )

Exposes the number of block ciphers available.

##### Returns

The number of available block ciphers (at least one).

##### Remarks

This is for use in enumerating block ciphers.

#### 6.17.2.3 ORDO\_PUBLIC const struct BLOCK\_CIPHER\* block\_cipher\_by\_name ( const char \* *name* )

Returns a block cipher primitive from a name.

## Parameters

<i>name</i>	A block cipher name.
-------------	----------------------

## Returns

The block cipher such that the following is true:

```
block_cipher_name(retval) = name
```

or 0 if no such block cipher exists.

#### 6.17.2.4 ORDO\_PUBLIC const struct BLOCK\_CIPHER\* block\_cipher\_by\_index ( size\_t *index* )

Returns a block cipher primitive from an index.

## Parameters

<i>in</i>	<i>index</i>	A block cipher index.
-----------	--------------	-----------------------

## Returns

The block cipher corresponding to the provided index, or 0 if no such block cipher exists.

## Remarks

Use `block_cipher_count()` to get an upper bound on block cipher indices (there will be at least one).

#### 6.17.2.5 ORDO\_PUBLIC void\* block\_cipher\_alloc ( const struct BLOCK\_CIPHER \* *primitive* )

Allocates a block cipher state.

## Parameters

<i>in</i>	<i>primitive</i>	A block cipher primitive.
-----------	------------------	---------------------------

## Returns

An allocated block cipher state, or 0 on error.

#### 6.17.2.6 ORDO\_PUBLIC int block\_cipher\_init ( const struct BLOCK\_CIPHER \* *primitive*, void \* *state*, const void \* *key*, size\_t *key\_len*, const void \* *params* )

Initializes a block cipher state.

## Parameters

<i>in</i>	<i>primitive</i>	A block cipher primitive.
<i>in, out</i>	<i>state</i>	An allocated block cipher state.
<i>in</i>	<i>key</i>	The cryptographic key to use.
<i>in</i>	<i>key_len</i>	The length, in bytes, of the key.
<i>in</i>	<i>params</i>	Block cipher specific parameters.

## Returns

`ORDO_SUCCESS` on success, else an error code.

6.17.2.7 ORDO\_PUBLIC void block\_cipher\_forward ( const struct BLOCK\_CIPHER \* *primitive*, const void \* *state*, void \* *block* )

Applies a block cipher's forward permutation.

**Parameters**

in	<i>primitive</i>	A block cipher primitive.
in	<i>state</i>	An initialized block cipher state.
in, out	<i>block</i>	A data block to permute.

**Remarks**

The block should be the size of the block cipher's block size.

**6.17.2.8** ORDO\_PUBLIC void block\_cipher\_inverse ( const struct BLOCK\_CIPHER \* *primitive*, const void \* *state*, void \* *block* )

Applies a block cipher's inverse permutation.

**Parameters**

in	<i>primitive</i>	A block cipher primitive.
in	<i>state</i>	An initialized block cipher state.
in, out	<i>block</i>	A data block to permute.

**Remarks**

The block should be the size of the block cipher's block size.

**6.17.2.9** ORDO\_PUBLIC void block\_cipher\_final ( const struct BLOCK\_CIPHER \* *primitive*, void \* *state* )

Finalizes a block cipher state.

**Parameters**

in	<i>primitive</i>	A block cipher primitive.
in, out	<i>state</i>	A block cipher state.

**6.17.2.10** ORDO\_PUBLIC void block\_cipher\_free ( const struct BLOCK\_CIPHER \* *primitive*, void \* *state* )

Frees a block cipher state.

**Parameters**

in	<i>primitive</i>	A block cipher primitive.
in, out	<i>state</i>	A block cipher state.

**6.17.2.11** ORDO\_PUBLIC void block\_cipher\_copy ( const struct BLOCK\_CIPHER \* *primitive*, void \* *dst*, const void \* *src* )

Copies a block cipher state to another.

**Parameters**

in	<i>primitive</i>	A block cipher primitive.
out	<i>dst</i>	The destination state.

<code>in</code>	<code>src</code>	The source state.
-----------------	------------------	-------------------

**Remarks**

Both states must have been initialized with the same block cipher and parameters, or this function's behaviour is undefined.

**6.17.2.12** `ORDO_PUBLIC size_t block_cipher_query ( const struct BLOCK_CIPHER * primitive, int query, size_t value )`

Queries a block cipher for suitable parameters.

**Parameters**

<code>in</code>	<code><i>primitive</i></code>	A block cipher primitive.
<code>in</code>	<code><i>query</i></code>	A query code.
<code>in</code>	<code><i>value</i></code>	A suggested value.

**Returns**

A suitable parameter of type `query` based on `value`.

**See Also**

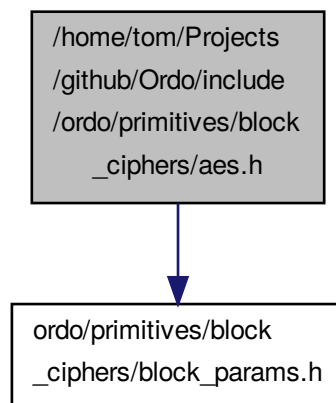
[query.h](#)

## 6.18 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_ciphers/aes.h File Reference

Primitive.

```
#include "ordo/primitives/block_ciphers/block_params.h"
```

Include dependency graph for aes.h:



## Functions

- ORDO\_PUBLIC struct AES\_STATE \* [aes\\_alloc](#) (void)
- ORDO\_PUBLIC int [aes\\_init](#) (struct AES\_STATE \*state, const void \*key, size\_t key\_len, const struct [AES\\_PARAMS](#) \*params)
- ORDO\_PUBLIC void [aes\\_forward](#) (const struct AES\_STATE \*state, uint8\_t \*block)
- ORDO\_PUBLIC void [aes\\_inverse](#) (const struct AES\_STATE \*state, uint8\_t \*block)
- ORDO\_PUBLIC void [aes\\_final](#) (struct AES\_STATE \*state)
- ORDO\_PUBLIC void [aes\\_free](#) (struct AES\_STATE \*state)
- ORDO\_PUBLIC void [aes\\_copy](#) (struct AES\_STATE \*dst, const struct AES\_STATE \*src)
- ORDO\_PUBLIC size\_t [aes\\_query](#) (int query, size\_t value)

### 6.18.1 Detailed Description

Primitive. AES (Advanced Encryption Standard) is a block cipher. It has a 128-bit block size and three possible key sizes, namely 128, 192 and 256 bits. It is based on the Rijndael cipher and was selected as the official encryption standard on November 2001 (FIPS 197).

### 6.18.2 Function Documentation

#### 6.18.2.1 ORDO\_PUBLIC struct AES\_STATE\* [aes\\_alloc](#) ( void )

See Also

[block\\_cipher\\_alloc\(\)](#)

#### 6.18.2.2 ORDO\_PUBLIC int [aes\\_init](#) ( struct AES\_STATE \* *state*, const void \* *key*, size\_t *key\_len*, const struct [AES\\_PARAMS](#) \* *params* )

See Also

[block\\_cipher\\_init\(\)](#)

Return values

<a href="#">ORDO_KEY_LEN</a>	if the key length is not 16, 24, or 32 (bytes).
<a href="#">ORDO_ARG</a>	if parameters were provided and requested zero rounds or more than 20 rounds.

#### 6.18.2.3 ORDO\_PUBLIC void [aes\\_forward](#) ( const struct AES\_STATE \* *state*, uint8\_t \* *block* )

See Also

[block\\_cipher\\_forward\(\)](#)

#### 6.18.2.4 ORDO\_PUBLIC void [aes\\_inverse](#) ( const struct AES\_STATE \* *state*, uint8\_t \* *block* )

See Also

[block\\_cipher\\_inverse\(\)](#)

6.18.2.5 ORDO\_PUBLIC void aes\_final ( struct AES\_STATE \* state )

See Also

[block\\_cipher\\_final\(\)](#)

6.18.2.6 ORDO\_PUBLIC void aes\_free ( struct AES\_STATE \* state )

See Also

[block\\_cipher\\_free\(\)](#)

6.18.2.7 ORDO\_PUBLIC void aes\_copy ( struct AES\_STATE \* dst, const struct AES\_STATE \* src )

See Also

[block\\_cipher\\_copy\(\)](#)

6.18.2.8 ORDO\_PUBLIC size\_t aes\_query ( int query, size\_t value )

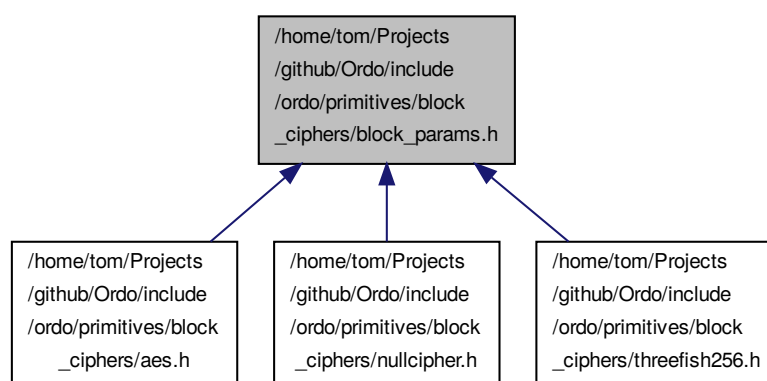
See Also

[block\\_cipher\\_query\(\)](#)

## 6.19 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_ciphers/block\_params.h File Reference

Primitive Parameters.

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



### Data Structures

- struct [THREEFISH256\\_PARAMS](#)  
*Threefish-256 block cipher parameters.*

- struct [AES\\_PARAMS](#)

*AES block cipher parameters.*

### 6.19.1 Detailed Description

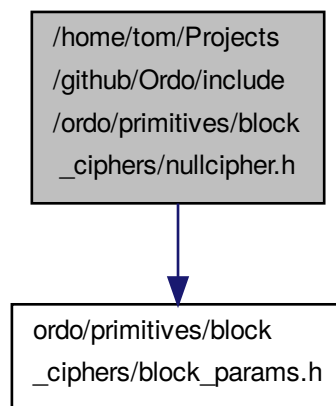
Primitive Parameters. This header contains parameter structures for all block ciphers.

## 6.20 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_ciphers/nullcipher.h File Reference

Primitive.

```
#include "ordo/primitives/block_ciphers/block_params.h"
```

Include dependency graph for nullcipher.h:



### Functions

- ORDO\_PUBLIC struct NULLCIPHER\_STATE \* [nullcipher\\_alloc](#) (void)
- ORDO\_PUBLIC int [nullcipher\\_init](#) (struct NULLCIPHER\_STATE \*state, const void \*key, size\_t key\_len, const void \*params)
- ORDO\_PUBLIC void [nullcipher\\_forward](#) (const struct NULLCIPHER\_STATE \*state, void \*block)
- ORDO\_PUBLIC void [nullcipher\\_inverse](#) (const struct NULLCIPHER\_STATE \*state, void \*block)
- ORDO\_PUBLIC void [nullcipher\\_final](#) (struct NULLCIPHER\_STATE \*state)
- ORDO\_PUBLIC void [nullcipher\\_free](#) (struct NULLCIPHER\_STATE \*state)
- ORDO\_PUBLIC void [nullcipher\\_copy](#) (struct NULLCIPHER\_STATE \*dst, const struct NULLCIPHER\_STATE \*src)
- ORDO\_PUBLIC size\_t [nullcipher\\_query](#) (int query, size\_t value)



## 6.20.1 Detailed Description

Primitive. This cipher is only used to debug the library and does absolutely nothing, in other words, it is the identity permutation. It accepts no key, that is it only accepts a key length of zero bytes. Its block size is 128 bits and is arbitrarily chosen.

## 6.20.2 Function Documentation

6.20.2.1 **ORDO\_PUBLIC** struct NULLCIPHER\_STATE\* nullcipher\_alloc ( void )

See Also

[block\\_cipher\\_alloc\(\)](#)

6.20.2.2 **ORDO\_PUBLIC** int nullcipher\_init ( struct NULLCIPHER\_STATE \* *state*, const void \* *key*, size\_t *key\_len*, const void \* *params* )

See Also

[block\\_cipher\\_init\(\)](#)

Return values

<a href="#">ORDO_KEY_LEN</a>	if the key length is not zero.
------------------------------	--------------------------------

6.20.2.3 **ORDO\_PUBLIC** void nullcipher\_forward ( const struct NULLCIPHER\_STATE \* *state*, void \* *block* )

See Also

[block\\_cipher\\_forward\(\)](#)

6.20.2.4 **ORDO\_PUBLIC** void nullcipher\_inverse ( const struct NULLCIPHER\_STATE \* *state*, void \* *block* )

See Also

[block\\_cipher\\_inverse\(\)](#)

6.20.2.5 **ORDO\_PUBLIC** void nullcipher\_final ( struct NULLCIPHER\_STATE \* *state* )

See Also

[block\\_cipher\\_final\(\)](#)

6.20.2.6 **ORDO\_PUBLIC** void nullcipher\_free ( struct NULLCIPHER\_STATE \* *state* )

See Also

[block\\_cipher\\_free\(\)](#)

6.20.2.7 **ORDO\_PUBLIC** void nullcipher\_copy ( struct NULLCIPHER\_STATE \* *dst*, const struct NULLCIPHER\_STATE \* *src* )

See Also

[block\\_cipher\\_copy\(\)](#)

6.20.2.8 ORDO\_PUBLIC size\_t nullcipher\_query ( int query, size\_t value )

See Also

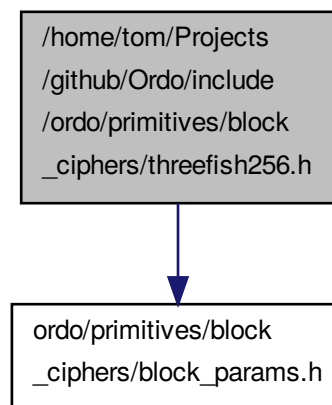
[block\\_cipher\\_query\(\)](#)

## 6.21 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_ciphers/threefish256.h File Reference

Primitive.

```
#include "ordo/primitives/block_ciphers/block_params.h"
```

Include dependency graph for threefish256.h:



### Functions

- ORDO\_PUBLIC struct THREEFISH256\_STATE \* [threefish256\\_alloc](#) (void)
- ORDO\_PUBLIC int [threefish256\\_init](#) (struct THREEFISH256\_STATE \*state, const uint64\_t \*key, size\_t key\_len, const struct [THREEFISH256\\_PARAMS](#) \*params)
- ORDO\_PUBLIC void [threefish256\\_forward](#) (const struct THREEFISH256\_STATE \*state, uint64\_t \*block)
- ORDO\_PUBLIC void [threefish256\\_inverse](#) (const struct THREEFISH256\_STATE \*state, uint64\_t \*block)
- ORDO\_PUBLIC void [threefish256\\_final](#) (struct THREEFISH256\_STATE \*state)
- ORDO\_PUBLIC void [threefish256\\_free](#) (struct THREEFISH256\_STATE \*state)
- ORDO\_PUBLIC void [threefish256\\_copy](#) (struct THREEFISH256\_STATE \*dst, const struct THREEFISH256\_STATE \*src)
- ORDO\_PUBLIC size\_t [threefish256\\_query](#) (int query, size\_t value)

### 6.21.1 Detailed Description

Primitive. Threefish-256 is a block cipher with a 256-bit block size and a 256-bit key size. It also has an optional 128-bit tweak, which can be set through the cipher parameters.

The Threefish ciphers were originally designed to be used as a building block for the Skein hash function family.

## 6.21.2 Function Documentation

6.21.2.1 **ORDO\_PUBLIC** struct THREEFISH256\_STATE\* threefish256\_alloc ( void )

See Also

[block\\_cipher\\_alloc\(\)](#)

6.21.2.2 **ORDO\_PUBLIC** int threefish256\_init ( struct THREEFISH256\_STATE \* *state*, const uint64\_t \* *key*, size\_t *key\_len*, const struct THREEFISH256\_PARAMS \* *params* )

See Also

[block\\_cipher\\_init\(\)](#)

Return values

<a href="#">ORDO_KEY_LEN</a>	if the key length is not 32 (bytes).
------------------------------	--------------------------------------

6.21.2.3 **ORDO\_PUBLIC** void threefish256\_forward ( const struct THREEFISH256\_STATE \* *state*, uint64\_t \* *block* )

See Also

[block\\_cipher\\_forward\(\)](#)

6.21.2.4 **ORDO\_PUBLIC** void threefish256\_inverse ( const struct THREEFISH256\_STATE \* *state*, uint64\_t \* *block* )

See Also

[block\\_cipher\\_inverse\(\)](#)

6.21.2.5 **ORDO\_PUBLIC** void threefish256\_final ( struct THREEFISH256\_STATE \* *state* )

See Also

[block\\_cipher\\_final\(\)](#)

6.21.2.6 **ORDO\_PUBLIC** void threefish256\_free ( struct THREEFISH256\_STATE \* *state* )

See Also

[block\\_cipher\\_free\(\)](#)

6.21.2.7 **ORDO\_PUBLIC** void threefish256\_copy ( struct THREEFISH256\_STATE \* *dst*, const struct THREEFISH256\_STATE \* *src* )

See Also

[block\\_cipher\\_copy\(\)](#)

6.21.2.8 **ORDO\_PUBLIC** size\_t threefish256\_query ( int *query*, size\_t *value* )

See Also

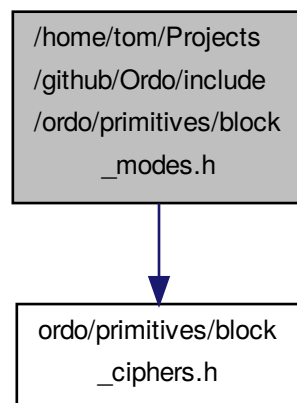
[block\\_cipher\\_query\(\)](#)

## 6.22 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_modes.h File Reference

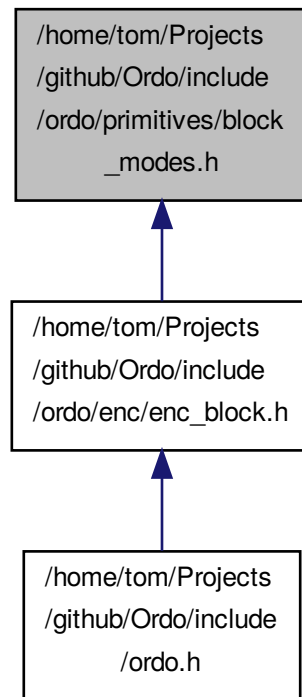
Abstraction Layer.

```
#include "ordo/primitives/block_ciphers.h"
```

Include dependency graph for block\_modes.h:



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



## Functions

- ORDO\_PUBLIC const char \* [block\\_mode\\_name](#) (const struct BLOCK\_MODE \*mode)
- ORDO\_PUBLIC const struct BLOCK\_MODE \* [ecb](#) (void)  
*The ECB (Electronic CodeBook) block mode of operation.*
- ORDO\_PUBLIC const struct BLOCK\_MODE \* [cbc](#) (void)  
*The CBC (Ciphertext Block Chaining) block mode of operation.*
- ORDO\_PUBLIC const struct BLOCK\_MODE \* [ctr](#) (void)  
*The CTR (Counter) block mode of operation.*
- ORDO\_PUBLIC const struct BLOCK\_MODE \* [cfb](#) (void)  
*The CFB (Cipher FeedBack) block mode of operation.*
- ORDO\_PUBLIC const struct BLOCK\_MODE \* [ofb](#) (void)  
*The OFB (Output FeedBack) block mode of operation.*
- ORDO\_PUBLIC size\_t [block\\_mode\\_count](#) (void)
- ORDO\_PUBLIC const struct BLOCK\_MODE \* [block\\_mode\\_by\\_name](#) (const char \*name)
- ORDO\_PUBLIC const struct BLOCK\_MODE \* [block\\_mode\\_by\\_index](#) (size\_t index)

- ORDO\_PUBLIC void \* [block\\_mode\\_alloc](#) (const struct BLOCK\_MODE \*mode, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC int [block\\_mode\\_init](#) (const struct BLOCK\_MODE \*mode, void \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const void \*iv, size\_t iv\_len, int direction, const void \*params)
- ORDO\_PUBLIC void [block\\_mode\\_update](#) (const struct BLOCK\_MODE \*mode, void \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const void \*in, size\_t in\_len, void \*out, size\_t out\_len)
- ORDO\_PUBLIC int [block\\_mode\\_final](#) (const struct BLOCK\_MODE \*mode, void \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, void \*out, size\_t out\_len)
- ORDO\_PUBLIC void [block\\_mode\\_free](#) (const struct BLOCK\_MODE \*mode, void \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC void [block\\_mode\\_copy](#) (const struct BLOCK\_MODE \*mode, const struct BLOCK\_CIPHER \*cipher, void \*dst, const void \*src)
- ORDO\_PUBLIC size\_t [block\\_mode\\_query](#) (const struct BLOCK\_MODE \*mode, const struct BLOCK\_CIPHER \*cipher, int query, size\_t value)

### 6.22.1 Detailed Description

Abstraction Layer. This abstraction layer declares all the block modes of operation in the library, making them available to higher level modules.

Note "block cipher mode of operation" is shortened to "block mode" in code and documentation to minimize noise and redundancy.

### 6.22.2 Function Documentation

#### 6.22.2.1 ORDO\_PUBLIC const char\* block\_mode\_name ( const struct BLOCK\_MODE \* mode )

Returns the name of a block mode primitive.

##### Parameters

in	<i>mode</i>	A block mode primitive.
----	-------------	-------------------------

##### Returns

Returns the block mode's name.

##### Remarks

This name can then be used in [block\\_mode\\_by\\_name\(\)](#).

#### 6.22.2.2 ORDO\_PUBLIC size\_t block\_mode\_count ( void )

Exposes the number of block modes available.

##### Returns

The number of available block modes (at least one).

##### Remarks

This is for use in enumerating block modes.

#### 6.22.2.3 ORDO\_PUBLIC const struct BLOCK\_MODE\* block\_mode\_by\_name ( const char \* name )

Returns a block mode primitive from a name.

## Parameters

<i>name</i>	A block mode name.
-------------	--------------------

## Returns

The block mode such that the following is true:

```
block_mode_name(retval) = name
```

or 0 if no such block mode exists.

6.22.2.4 ORDO\_PUBLIC const struct BLOCK\_MODE\* block\_mode\_by\_index ( size\_t *index* )

Returns a block cipher mode from an index.

## Parameters

<i>in</i>	<i>index</i>	A block mode index.
-----------	--------------	---------------------

## Returns

The block mode corresponding to the provided index, or 0 if no such block mode exists.

## Remarks

Use `block_mode_count()` to get an upper bound on the block mode indices (there will be at least one).

6.22.2.5 ORDO\_PUBLIC void\* block\_mode\_alloc ( const struct BLOCK\_MODE \* *mode*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

Allocates a block mode state.

## Parameters

<i>in</i>	<i>mode</i>	A block mode primitive.
<i>in</i>	<i>cipher</i>	A block cipher primitive.
<i>in</i>	<i>cipher_state</i>	An allocated block cipher state.

## Returns

An allocated block mode state, or 0 on error.

6.22.2.6 ORDO\_PUBLIC int block\_mode\_init ( const struct BLOCK\_MODE \* *mode*, void \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const void \* *iv*, size\_t *iv\_len*, int *direction*, const void \* *params* )

Initializes a block mode state.

## Parameters

<i>in</i>	<i>mode</i>	A block mode primitive.
<i>in, out</i>	<i>state</i>	A block mode state.

in	<i>cipher</i>	A block cipher primitive.
in	<i>cipher_state</i>	A block cipher state.
in	<i>iv</i>	The initialization vector to use.
in	<i>iv_len</i>	The length, in bytes, of the IV.
in	<i>direction</i>	1 for encryption, 0 for decryption.
in	<i>params</i>	Block mode specific parameters.

**Returns**

[ORDO\\_SUCCESS](#) on success, else an error code.

**6.22.2.7** `ORDO_PUBLIC void block_mode_update ( const struct BLOCK_MODE * mode, void * state, const struct BLOCK_CIPHER * cipher, const void * cipher_state, const void * in, size_t in_len, void * out, size_t * out_len )`

Encrypts or decrypts a buffer.

**Parameters**

in	<i>mode</i>	A block mode primitive.
in, out	<i>state</i>	A block mode state.
in	<i>cipher</i>	A block cipher primitive.
in	<i>cipher_state</i>	A block cipher state.
in	<i>in</i>	The input buffer.
in	<i>in_len</i>	The length, in bytes, of the input.
out	<i>out</i>	The output buffer.
out	<i>out_len</i>	A pointer to an integer to which to write the number of output bytes that can be returned to the user. Remaining input data has <b>not</b> been ignored and should not be passed again.

**Remarks**

In-place encryption (by letting *in* be the same buffer as *out*) may not be supported by *mode*, check the documentation.

**6.22.2.8** `ORDO_PUBLIC int block_mode_final ( const struct BLOCK_MODE * mode, void * state, const struct BLOCK_CIPHER * cipher, const void * cipher_state, void * out, size_t * out_len )`

Finalizes a block mode state.

**Parameters**

in	<i>mode</i>	A block mode primitive.
in, out	<i>state</i>	A block mode state.
in	<i>cipher</i>	A block cipher primitive.
in	<i>cipher_state</i>	A block cipher state.
out	<i>out</i>	The output buffer.
out	<i>out_len</i>	A pointer to an integer to which to store the number of bytes written to <i>out</i> .

**Returns**

[ORDO\\_SUCCESS](#) on success, else an error code.

**Remarks**

This function will return any input bytes which were not returned by calls to `block_mode_update()` (in the correct order).



6.22.2.9 ORDO\_PUBLIC void block\_mode\_free ( const struct BLOCK\_MODE \* *mode*, void \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

Frees a block mode state.

**Parameters**

in	<i>mode</i>	A block mode primitive.
in, out	<i>state</i>	A block mode state.
in	<i>cipher</i>	A block cipher primitive.
in	<i>cipher_state</i>	A block cipher state.

**6.22.2.10** `ORDO_PUBLIC void block_mode_copy ( const struct BLOCK_MODE * mode, const struct BLOCK_CIPHER * cipher, void * dst, const void * src )`

Copies a block mode state to another.

**Parameters**

in	<i>mode</i>	A block mode primitive.
in	<i>cipher</i>	A block cipher primitive.
out	<i>dst</i>	The destination state.
out	<i>src</i>	The source state.

**Remarks**

Both states must have been initialized with the same block mode, block cipher, and parameters (for both).

**6.22.2.11** `ORDO_PUBLIC size_t block_mode_query ( const struct BLOCK_MODE * mode, const struct BLOCK_CIPHER * cipher, int query, size_t value )`

Queries a block mode for suitable parameters.

**Parameters**

in	<i>mode</i>	A block mode primitive.
in	<i>cipher</i>	A block cipher primitive.
in	<i>query</i>	A query code.
in	<i>value</i>	A suggested value.

**Returns**

A suitable parameter of type `query` based on `value`.

**See Also**

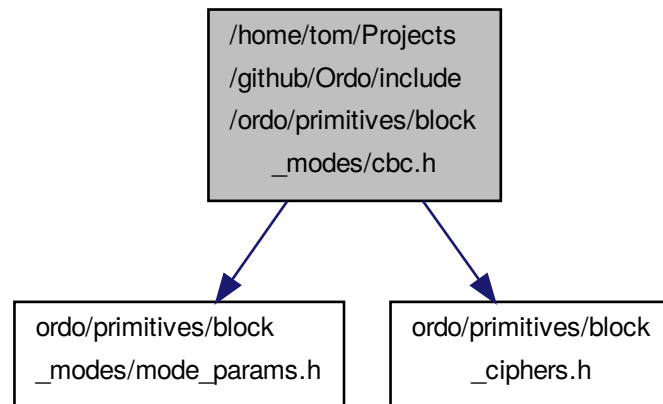
[query.h](#)

## 6.23 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_modes/cbc.h File Reference

**Primitive.**

```
#include "ordo/primitives/block_modes/mode_params.h"
#include "ordo/primitives/block_ciphers.h"
```

Include dependency graph for cbc.h:



## Functions

- ORDO\_PUBLIC struct CBC\_STATE \* [cbc\\_alloc](#) (const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC int [cbc\\_init](#) (struct CBC\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const void \*iv, size\_t iv\_len, int dir, const struct CBC\_PARAMS \*params)
- ORDO\_PUBLIC void [cbc\\_update](#) (struct CBC\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const unsigned char \*in, size\_t in\_len, unsigned char \*out, size\_t \*out\_len)
- ORDO\_PUBLIC int [cbc\\_final](#) (struct CBC\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, unsigned char \*out, size\_t \*out\_len)
- ORDO\_PUBLIC void [cbc\\_free](#) (struct CBC\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC void [cbc\\_copy](#) (struct CBC\_STATE \*dst, const struct CBC\_STATE \*src, const struct BLOCK\_CIPHER \*cipher)
- ORDO\_PUBLIC size\_t [cbc\\_query](#) (const struct BLOCK\_CIPHER \*cipher, int query, size\_t value)

### 6.23.1 Detailed Description

Primitive. The CBC mode divides the input message into blocks of the cipher's block size, and encrypts them in a sequential fashion, where each block depends on the previous one (and the first block depends on the initialization vector). If the input message's length is not a multiple of the cipher's block size, a padding mechanism is enabled by default which will pad the message to the correct length (and remove the extra data upon decryption). If padding is explicitly disabled through the mode of operation's parameters, the input's length must be a multiple of the cipher's block size.

If padding is enabled, [cbc\\_final\(\)](#) requires a valid pointer to be passed in the `outlen` parameter and will always return a full blocksize of data, containing the last few ciphertext bytes containing the padding information.

If padding is disabled, `outlen` is also required, and will return the number of unprocessed plaintext bytes in the context. If this is any value other than zero, the function will also fail with `ORDO_LEFTOVER`.

### 6.23.2 Function Documentation

6.23.2.1 ORDO\_PUBLIC struct CBC\_STATE\* cbc\_alloc ( const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

See Also

[block\\_mode\\_alloc\(\)](#)

6.23.2.2 ORDO\_PUBLIC int cbc\_init ( struct CBC\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const void \* *iv*, size\_t *iv\_len*, int *dir*, const struct CBC\_PARAMS \* *params* )

See Also

[block\\_mode\\_init\(\)](#)

6.23.2.3 ORDO\_PUBLIC void cbc\_update ( struct CBC\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const unsigned char \* *in*, size\_t *in\_len*, unsigned char \* *out*, size\_t \* *out\_len* )

See Also

[block\\_mode\\_update\(\)](#)

6.23.2.4 ORDO\_PUBLIC int cbc\_final ( struct CBC\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, unsigned char \* *out*, size\_t \* *out\_len* )

See Also

[block\\_mode\\_final\(\)](#)

6.23.2.5 ORDO\_PUBLIC void cbc\_free ( struct CBC\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

See Also

[block\\_mode\\_free\(\)](#)

6.23.2.6 ORDO\_PUBLIC void cbc\_copy ( struct CBC\_STATE \* *dst*, const struct CBC\_STATE \* *src*, const struct BLOCK\_CIPHER \* *cipher* )

See Also

[block\\_mode\\_copy\(\)](#)

6.23.2.7 ORDO\_PUBLIC size\_t cbc\_query ( const struct BLOCK\_CIPHER \* *cipher*, int *query*, size\_t *value* )

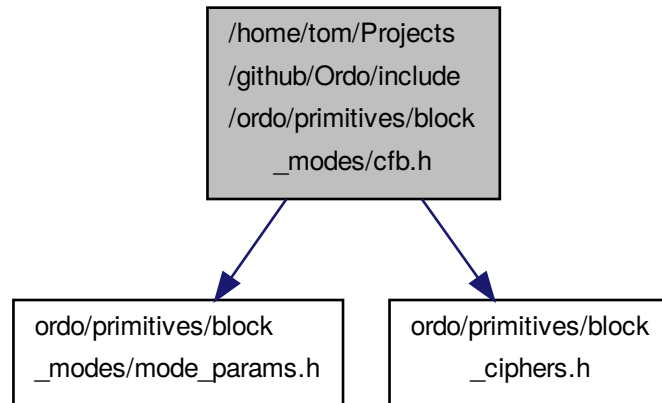
See Also

[block\\_mode\\_query\(\)](#)

## 6.24 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_modes/cfb.h File Reference

Primitive.

```
#include "ordo/primitives/block_modes/mode_params.h"
#include "ordo/primitives/block_ciphers.h"
Include dependency graph for cfb.h:
```



## Functions

- ORDO\_PUBLIC struct CFB\_STATE \* [cfb\\_alloc](#) (const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC int [cfb\\_init](#) (struct CFB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const void \*iv, size\_t iv\_len, int dir, const void \*params)
- ORDO\_PUBLIC void [cfb\\_update](#) (struct CFB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const unsigned char \*in, size\_t in\_len, unsigned char \*out, size\_t out\_len)
- ORDO\_PUBLIC int [cfb\\_final](#) (struct CFB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, unsigned char \*out, size\_t out\_len)
- ORDO\_PUBLIC void [cfb\\_free](#) (struct CFB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC void [cfb\\_copy](#) (struct CFB\_STATE \*dst, const struct CFB\_STATE \*src, const struct BLOCK\_CIPHER \*cipher)
- ORDO\_PUBLIC size\_t [cfb\\_query](#) (const struct BLOCK\_CIPHER \*cipher, int query, size\_t value)

### 6.24.1 Detailed Description

Primitive. The CFB mode generates a keystream by repeatedly encrypting an initialization vector and mixing in the plaintext, effectively turning a block cipher into a stream cipher. As such, CFB mode requires no padding, and the ciphertext size will always be equal to the plaintext size.

Note that the CFB keystream depends on the plaintext fed into it, as opposed to OFB mode. This also means the block cipher's inverse permutation is never used.

[cfb\\_final\(\)](#) accepts 0 as an argument for `outlen`, since by design the CFB mode of operation does not produce any final data. However, if a valid pointer is passed, its value will be set to zero as expected.

### 6.24.2 Function Documentation

6.24.2.1 ORDO\_PUBLIC struct CFB\_STATE\* [cfb\\_alloc](#) ( const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

See Also

[block\\_mode\\_alloc\(\)](#)

6.24.2.2 ORDO\_PUBLIC int cfb\_init ( struct CFB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const void \* *iv*, size\_t *iv\_len*, int *dir*, const void \* *params* )

See Also

[block\\_mode\\_init\(\)](#)

6.24.2.3 ORDO\_PUBLIC void cfb\_update ( struct CFB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const unsigned char \* *in*, size\_t *in\_len*, unsigned char \* *out*, size\_t \* *out\_len* )

See Also

[block\\_mode\\_update\(\)](#)

6.24.2.4 ORDO\_PUBLIC int cfb\_final ( struct CFB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, unsigned char \* *out*, size\_t \* *out\_len* )

See Also

[block\\_mode\\_final\(\)](#)

6.24.2.5 ORDO\_PUBLIC void cfb\_free ( struct CFB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

See Also

[block\\_mode\\_free\(\)](#)

6.24.2.6 ORDO\_PUBLIC void cfb\_copy ( struct CFB\_STATE \* *dst*, const struct CFB\_STATE \* *src*, const struct BLOCK\_CIPHER \* *cipher* )

See Also

[block\\_mode\\_copy\(\)](#)

6.24.2.7 ORDO\_PUBLIC size\_t cfb\_query ( const struct BLOCK\_CIPHER \* *cipher*, int *query*, size\_t *value* )

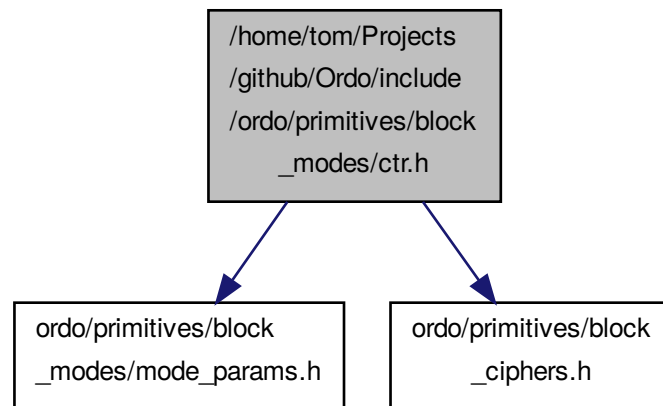
See Also

[block\\_mode\\_query\(\)](#)

## 6.25 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_modes/ctr.h File Reference

Primitive.

```
#include "ordo/primitives/block_modes/mode_params.h"
#include "ordo/primitives/block_ciphers.h"
Include dependency graph for ctr.h:
```



## Functions

- ORDO\_PUBLIC struct CTR\_STATE \* [ctr\\_alloc](#) (const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC int [ctr\\_init](#) (struct CTR\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const void \*iv, size\_t iv\_len, int dir, const void \*params)
- ORDO\_PUBLIC void [ctr\\_update](#) (struct CTR\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const unsigned char \*in, size\_t in\_len, unsigned char \*out, size\_t \*out\_len)
- ORDO\_PUBLIC int [ctr\\_final](#) (struct CTR\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, unsigned char \*out, size\_t \*out\_len)
- ORDO\_PUBLIC void [ctr\\_free](#) (struct CTR\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC void [ctr\\_copy](#) (struct CTR\_STATE \*dst, const struct CTR\_STATE \*src, const struct BLOCK\_CIPHER \*cipher)
- ORDO\_PUBLIC size\_t [ctr\\_query](#) (const struct BLOCK\_CIPHER \*cipher, int query, size\_t value)

### 6.25.1 Detailed Description

Primitive. The CTR mode generates a keystream by repeatedly encrypting a counter starting from some initialization vector, effectively turning a block cipher into a stream cipher. As such, CTR mode requires no padding, and outlen will always be equal to inlen.

Note that the CTR keystream is independent of the plaintext, and is also spatially coherent (using a given initialization vector on a len-byte message will "use up" len bytes of the keystream) so care must be taken to avoid reusing the initialization vector in an insecure way. This also means the block cipher's inverse permutation is never used.

[ctr\\_final\(\)](#) accepts 0 as an argument for `outlen`, since by design the CTR mode of operation does not produce any final data. However, if a valid pointer is passed, its value will be set to zero as expected.

### 6.25.2 Function Documentation

6.25.2.1 ORDO\_PUBLIC struct CTR\_STATE\* ctr\_alloc ( const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

See Also

[block\\_mode\\_alloc\(\)](#)

6.25.2.2 ORDO\_PUBLIC int ctr\_init ( struct CTR\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const void \* *iv*, size\_t *iv\_len*, int *dir*, const void \* *params* )

See Also

[block\\_mode\\_init\(\)](#)

6.25.2.3 ORDO\_PUBLIC void ctr\_update ( struct CTR\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const unsigned char \* *in*, size\_t *in\_len*, unsigned char \* *out*, size\_t \* *out\_len* )

See Also

[block\\_mode\\_update\(\)](#)

6.25.2.4 ORDO\_PUBLIC int ctr\_final ( struct CTR\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, unsigned char \* *out*, size\_t \* *out\_len* )

See Also

[block\\_mode\\_final\(\)](#)

6.25.2.5 ORDO\_PUBLIC void ctr\_free ( struct CTR\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

See Also

[block\\_mode\\_free\(\)](#)

6.25.2.6 ORDO\_PUBLIC void ctr\_copy ( struct CTR\_STATE \* *dst*, const struct CTR\_STATE \* *src*, const struct BLOCK\_CIPHER \* *cipher* )

See Also

[block\\_mode\\_copy\(\)](#)

6.25.2.7 ORDO\_PUBLIC size\_t ctr\_query ( const struct BLOCK\_CIPHER \* *cipher*, int *query*, size\_t *value* )

See Also

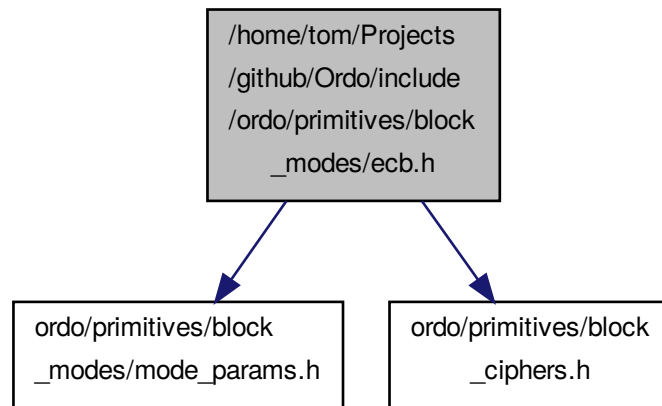
[block\\_mode\\_query\(\)](#)

## 6.26 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_modes/ecb.h File Reference

Primitive.



```
#include "ordo/primitives/block_modes/mode_params.h"
#include "ordo/primitives/block_ciphers.h"
Include dependency graph for ecb.h:
```



## Functions

- ORDO\_PUBLIC struct ECB\_STATE \* [ecb\\_alloc](#) (const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC int [ecb\\_init](#) (struct ECB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const void \*iv, size\_t iv\_len, int dir, const struct [ECB\\_PARAMS](#) \*params)
- ORDO\_PUBLIC void [ecb\\_update](#) (struct ECB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const unsigned char \*in, size\_t in\_len, unsigned char \*out, size\_t \*out\_len)
- ORDO\_PUBLIC int [ecb\\_final](#) (struct ECB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, unsigned char \*out, size\_t \*out\_len)
- ORDO\_PUBLIC void [ecb\\_free](#) (struct ECB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC void [ecb\\_copy](#) (struct ECB\_STATE \*dst, const struct ECB\_STATE \*src, const struct BLOCK\_CIPHER \*cipher)
- ORDO\_PUBLIC size\_t [ecb\\_query](#) (const struct BLOCK\_CIPHER \*cipher, int query, size\_t value)

### 6.26.1 Detailed Description

Primitive. The ECB mode divides the input message into blocks of the cipher's block size, and encrypts them individually and independently. If the input message's length is not a multiple of the cipher's block size, a padding mechanism is enabled by default which will pad the message to the correct length (and remove the extra data upon decryption). Padding may be disabled via [ECB\\_PARAMS](#), putting constraints on the input message.

The ECB mode does not require an initialization vector.

Note that the ECB mode is insecure in almost all situations and is not recommended for general purpose use.

### 6.26.2 Function Documentation

6.26.2.1 ORDO\_PUBLIC struct ECB\_STATE\* [ecb\\_alloc](#) ( const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

See Also

[block\\_mode\\_alloc\(\)](#)

6.26.2.2 **ORDO\_PUBLIC** int ecb\_init ( struct ECB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const void \* *iv*, size\_t *iv\_len*, int *dir*, const struct ECB\_PARAMS \* *params* )

See Also

[block\\_mode\\_init\(\)](#)

6.26.2.3 **ORDO\_PUBLIC** void ecb\_update ( struct ECB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const unsigned char \* *in*, size\_t *in\_len*, unsigned char \* *out*, size\_t \* *out\_len* )

See Also

[block\\_mode\\_update\(\)](#)

6.26.2.4 **ORDO\_PUBLIC** int ecb\_final ( struct ECB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, unsigned char \* *out*, size\_t \* *out\_len* )

See Also

[block\\_mode\\_final\(\)](#)

6.26.2.5 **ORDO\_PUBLIC** void ecb\_free ( struct ECB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

See Also

[block\\_mode\\_free\(\)](#)

6.26.2.6 **ORDO\_PUBLIC** void ecb\_copy ( struct ECB\_STATE \* *dst*, const struct ECB\_STATE \* *src*, const struct BLOCK\_CIPHER \* *cipher* )

See Also

[block\\_mode\\_copy\(\)](#)

6.26.2.7 **ORDO\_PUBLIC** size\_t ecb\_query ( const struct BLOCK\_CIPHER \* *cipher*, int *query*, size\_t *value* )

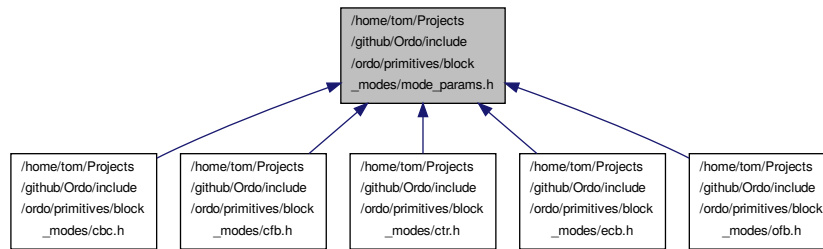
See Also

[block\\_mode\\_query\(\)](#)

## 6.27 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_modes/mode\_params.h File Reference

Primitive Parameters.

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



## Data Structures

- struct [ECB\\_PARAMS](#)  
*ECB parameters.*
- struct [CBC\\_PARAMS](#)  
*CBC parameters.*

### 6.27.1 Detailed Description

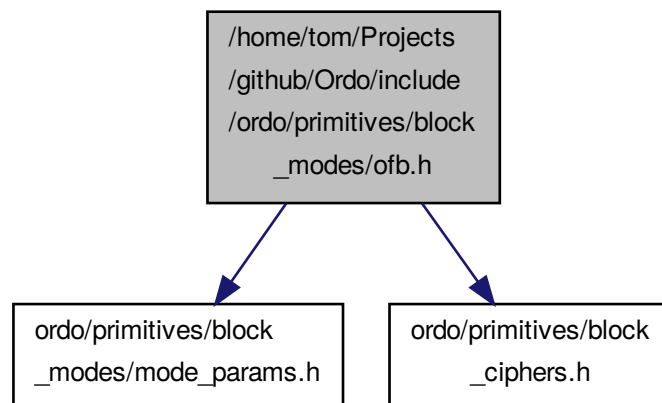
Primitive Parameters. This header contains parameter structures for all block modes.

## 6.28 /home/tom/Projects/github/Ordo/include/ordo/primitives/block\_modes/ofb.h File Reference

Primitive.

```
#include "ordo/primitives/block_modes/mode_params.h"
#include "ordo/primitives/block_ciphers.h"
```

Include dependency graph for ofb.h:



## Functions

- ORDO\_PUBLIC struct OFB\_STATE \* [ofb\\_alloc](#) (const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC int [ofb\\_init](#) (struct OFB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const void \*iv, size\_t iv\_len, int dir, const void \*params)
- ORDO\_PUBLIC void [ofb\\_update](#) (struct OFB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, const unsigned char \*in, size\_t in\_len, unsigned char \*out, size\_t \*out\_len)
- ORDO\_PUBLIC int [ofb\\_final](#) (struct OFB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state, unsigned char \*out, size\_t \*out\_len)
- ORDO\_PUBLIC void [ofb\\_free](#) (struct OFB\_STATE \*state, const struct BLOCK\_CIPHER \*cipher, const void \*cipher\_state)
- ORDO\_PUBLIC void [ofb\\_copy](#) (struct OFB\_STATE \*dst, const struct OFB\_STATE \*src, const struct BLOCK\_CIPHER \*cipher)
- ORDO\_PUBLIC size\_t [ofb\\_query](#) (const struct BLOCK\_CIPHER \*cipher, int query, size\_t value)

### 6.28.1 Detailed Description

Primitive. The OFB mode generates a keystream by repeatedly encrypting an initialization vector, effectively turning a block cipher into a stream cipher. As such, OFB mode requires no padding, and outlen will always be equal to inlen.

Note that the OFB keystream is independent of the plaintext, so a key/iv pair must never be used for more than one message. This also means the block cipher's inverse permutation is never used.

[ofb\\_final\(\)](#) accepts 0 as an argument for `outlen`, since by design the OFB mode of operation does not produce any final data. However, if a valid pointer is passed, its value will be set to zero as expected.

### 6.28.2 Function Documentation

6.28.2.1 ORDO\_PUBLIC struct OFB\_STATE\* [ofb\\_alloc](#) ( const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

See Also

[block\\_mode\\_alloc\(\)](#)

6.28.2.2 ORDO\_PUBLIC int [ofb\\_init](#) ( struct OFB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const void \* *iv*, size\_t *iv\_len*, int *dir*, const void \* *params* )

See Also

[block\\_mode\\_init\(\)](#)

6.28.2.3 ORDO\_PUBLIC void [ofb\\_update](#) ( struct OFB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, const unsigned char \* *in*, size\_t *in\_len*, unsigned char \* *out*, size\_t \* *out\_len* )

See Also

[block\\_mode\\_update\(\)](#)

6.28.2.4 ORDO\_PUBLIC int [ofb\\_final](#) ( struct OFB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state*, unsigned char \* *out*, size\_t \* *out\_len* )

See Also

[block\\_mode\\_final\(\)](#)

6.28.2.5 ORDO\_PUBLIC void ofb\_free ( struct OFB\_STATE \* *state*, const struct BLOCK\_CIPHER \* *cipher*, const void \* *cipher\_state* )

See Also

[block\\_mode\\_free\(\)](#)

6.28.2.6 ORDO\_PUBLIC void ofb\_copy ( struct OFB\_STATE \* *dst*, const struct OFB\_STATE \* *src*, const struct BLOCK\_CIPHER \* *cipher* )

See Also

[block\\_mode\\_copy\(\)](#)

6.28.2.7 ORDO\_PUBLIC size\_t ofb\_query ( const struct BLOCK\_CIPHER \* *cipher*, int *query*, size\_t *value* )

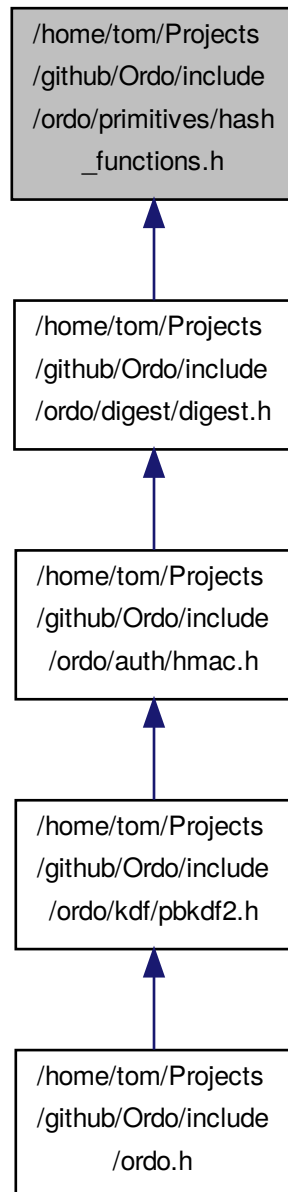
See Also

[block\\_mode\\_query\(\)](#)

## 6.29 /home/tom/Projects/github/Ordo/include/ordo/primitives/hash\_functions.h File Reference

Abstraction Layer.

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



## Functions

- ORDO\_PUBLIC const char \* [hash\\_function\\_name](#) (const struct HASH\_FUNCTION \*primitive)
- ORDO\_PUBLIC const struct HASH\_FUNCTION \* [sha256](#) (void)  
*The SHA-256 hash function.*
- ORDO\_PUBLIC const struct HASH\_FUNCTION \* [md5](#) (void)  
*The MD5 hash function.*

- ORDO\_PUBLIC const struct HASH\_FUNCTION \* [skein256](#) (void)  
*The Skein-256 hash function.*
- ORDO\_PUBLIC size\_t [hash\\_function\\_count](#) (void)
- ORDO\_PUBLIC const struct HASH\_FUNCTION \* [hash\\_function\\_by\\_name](#) (const char \*name)
- ORDO\_PUBLIC const struct HASH\_FUNCTION \* [hash\\_function\\_by\\_index](#) (size\_t index)
- ORDO\_PUBLIC void \* [hash\\_function\\_alloc](#) (const struct HASH\_FUNCTION \*primitive)
- ORDO\_PUBLIC int [hash\\_function\\_init](#) (const struct HASH\_FUNCTION \*primitive, void \*state, const void \*params)
- ORDO\_PUBLIC void [hash\\_function\\_update](#) (const struct HASH\_FUNCTION \*primitive, void \*state, const void \*buffer, size\_t len)
- ORDO\_PUBLIC void [hash\\_function\\_final](#) (const struct HASH\_FUNCTION \*primitive, void \*state, void \*digest)
- ORDO\_PUBLIC void [hash\\_function\\_free](#) (const struct HASH\_FUNCTION \*primitive, void \*state)
- ORDO\_PUBLIC void [hash\\_function\\_copy](#) (const struct HASH\_FUNCTION \*primitive, void \*dst, const void \*src)
- ORDO\_PUBLIC size\_t [hash\\_function\\_query](#) (const struct HASH\_FUNCTION \*primitive, int query, size\_t value)

### 6.29.1 Detailed Description

Abstraction Layer. This abstraction layer declares all the hash functions and also makes them available to higher level modules - for a slightly more convenient wrapper to this interface, you can use [digest.h](#).

### 6.29.2 Function Documentation

#### 6.29.2.1 ORDO\_PUBLIC const char\* hash\_function\_name ( const struct HASH\_FUNCTION \* *primitive* )

Returns the name of a hash function primitive.

##### Parameters

in	<i>primitive</i>	A hash function primitive.
----	------------------	----------------------------

##### Returns

Returns the hash function's name.

##### Remarks

This name can then be used in [hash\\_function\\_by\\_name\(\)](#).

#### 6.29.2.2 ORDO\_PUBLIC size\_t hash\_function\_count ( void )

Exposes the number of hash functions available.

##### Returns

The number of available hash functions (at least one).

##### Remarks

This is for use in enumerating hash functions.

6.29.2.3 ORDO\_PUBLIC `const struct HASH_FUNCTION*` `hash_function_by_name ( const char * name )`

Returns a hash function primitive from a name.



## Parameters

<i>name</i>	A hash function name.
-------------	-----------------------

## Returns

The hash function such that the following is true:

```
hash_function_name(retval) = name
```

or 0 if no such hash function exists.

6.29.2.4 ORDO\_PUBLIC const struct HASH\_FUNCTION\* hash\_function\_by\_index ( size\_t *index* )

Returns a hash function primitive from an index.

## Parameters

in	<i>index</i>	A hash function index.
----	--------------	------------------------

## Returns

The hash function corresponding to the provided index, or 0 if no such hash function exists.

## Remarks

Use `hash_function_count()` to obtain an upper bound on hash function indices (there will be at least one).

6.29.2.5 ORDO\_PUBLIC void\* hash\_function\_alloc ( const struct HASH\_FUNCTION \* *primitive* )

Allocates a hash function state.

## Parameters

in	<i>primitive</i>	A hash function primitive.
----	------------------	----------------------------

## Returns

An allocated hash function state, or 0 on error.

6.29.2.6 ORDO\_PUBLIC int hash\_function\_init ( const struct HASH\_FUNCTION \* *primitive*, void \* *state*, const void \* *params* )

Initializes a hash function state.

## Parameters

in	<i>primitive</i>	A hash function primitive.
in, out	<i>state</i>	An allocated hash function state.
in	<i>params</i>	Hash function specific parameters.

## Returns

`ORDO_SUCCESS` on success, else an error code.

6.29.2.7 ORDO\_PUBLIC void hash\_function\_update ( const struct HASH\_FUNCTION \* *primitive*, void \* *state*, const void \* *buffer*, size\_t *len* )

Updates a hash function state by appending a buffer to the message this state is to calculate the cryptographic digest of.

**Parameters**

in	<i>primitive</i>	A hash function primitive.
in, out	<i>state</i>	A hash function state.
in	<i>buffer</i>	A buffer to append to the message.
in	<i>len</i>	The length, in bytes, of the buffer.

**Remarks**

This function has the property that doing `update (x)` followed by `update (y)` is equivalent to `update (x || y)`, where `||` denotes concatenation.

#### 6.29.2.8 ORDO\_PUBLIC void hash\_function\_final ( const struct HASH\_FUNCTION \* *primitive*, void \* *state*, void \* *digest* )

Finalizes a hash function state, outputting the final digest.

**Parameters**

in	<i>primitive</i>	A hash function primitive.
in, out	<i>state</i>	A hash function state.
out	<i>digest</i>	A buffer in which to write the digest.

**Remarks**

The `digest` buffer should be as large as the hash function's digest length (unless you changed it via custom parameters).

#### 6.29.2.9 ORDO\_PUBLIC void hash\_function\_free ( const struct HASH\_FUNCTION \* *primitive*, void \* *state* )

Frees a hash function state.

**Parameters**

in	<i>primitive</i>	A hash function primitive.
in, out	<i>state</i>	A hash function state.

#### 6.29.2.10 ORDO\_PUBLIC void hash\_function\_copy ( const struct HASH\_FUNCTION \* *primitive*, void \* *dst*, const void \* *src* )

Copies a hash function state to another.

**Parameters**

in	<i>primitive</i>	A hash function primitive.
out	<i>dst</i>	The destination state.
in	<i>src</i>	The source state.

**Remarks**

The states must have been initialized with the same hash function and parameters, or this function's behaviour is undefined.

#### 6.29.2.11 ORDO\_PUBLIC size\_t hash\_function\_query ( const struct HASH\_FUNCTION \* *primitive*, int *query*, size\_t *value* )

Queries a hash function for suitable parameters.

## Parameters

in	<i>primitive</i>	A hash function primitive.
in	<i>query</i>	A query code.
in	<i>value</i>	A suggested value.

## Returns

A suitable parameter of type `query` based on `value`.

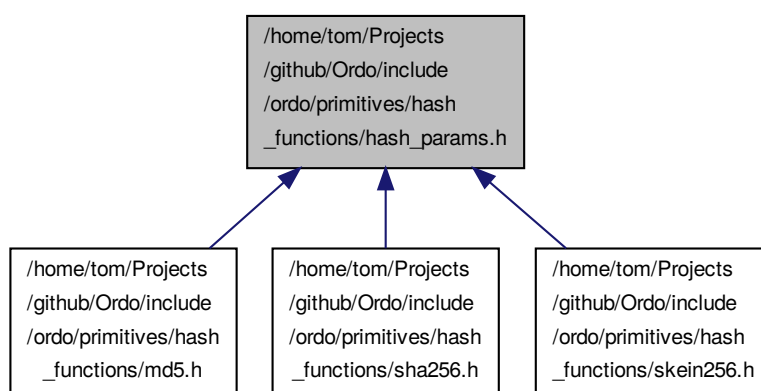
## See Also

[query.h](#)

## 6.30 /home/tom/Projects/github/Ordo/include/ordo/primitives/hash\_functions/hash\_params.h File Reference

Primitive Parameters.

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



## Data Structures

- struct [SKEIN256\\_PARAMS](#)  
*Skein-256 hash function parameters.*

## Functions

- ORDO\_PUBLIC struct [SKEIN256\\_PARAMS](#) [skein256\\_default](#) (void)  
*Returns the default Skein-256 configuration block (parameters).*

### 6.30.1 Detailed Description

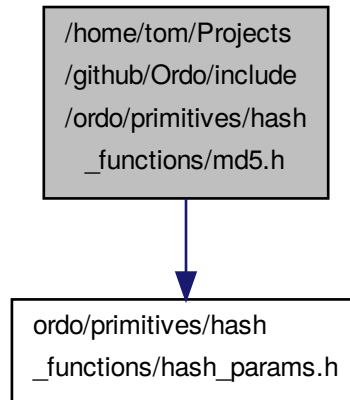
Primitive Parameters. This header contains parameter structures for all hash functions.

## 6.31 /home/tom/Projects/github/Ordo/include/ordo/primitives/hash\_functions/md5.h File Reference

Primitive.

```
#include "ordo/primitives/hash_functions/hash_params.h"
```

Include dependency graph for md5.h:



### Functions

- ORDO\_PUBLIC struct MD5\_STATE \* [md5\\_alloc](#) (void)
- ORDO\_PUBLIC int [md5\\_init](#) (struct MD5\_STATE \*state, const void \*params)
- ORDO\_PUBLIC void [md5\\_update](#) (struct MD5\_STATE \*state, const void \*buffer, size\_t len)
- ORDO\_PUBLIC void [md5\\_final](#) (struct MD5\_STATE \*state, void \*digest)
- ORDO\_PUBLIC void [md5\\_free](#) (struct MD5\_STATE \*state)
- ORDO\_PUBLIC void [md5\\_copy](#) (struct MD5\_STATE \*dst, const struct MD5\_STATE \*src)
- ORDO\_PUBLIC size\_t [md5\\_query](#) (int query, size\_t value)

#### 6.31.1 Detailed Description

Primitive. The MD5 hash function, which produces a 128-bit digest.

#### 6.31.2 Function Documentation

6.31.2.1 ORDO\_PUBLIC struct MD5\_STATE\* [md5\\_alloc](#) ( void )

See Also

[hash\\_function\\_alloc\(\)](#)

6.31.2.2 ORDO\_PUBLIC int [md5\\_init](#) ( struct MD5\_STATE \* *state*, const void \* *params* )

## See Also

`hash_function_init()`

## Remarks

The `params` parameter is ignored.

6.31.2.3 ORDO\_PUBLIC void md5\_update ( struct MD5\_STATE \* *state*, const void \* *buffer*, size\_t *len* )

## See Also

`hash_function_update()`

6.31.2.4 ORDO\_PUBLIC void md5\_final ( struct MD5\_STATE \* *state*, void \* *digest* )

## See Also

`hash_function_final()`

6.31.2.5 ORDO\_PUBLIC void md5\_free ( struct MD5\_STATE \* *state* )

## See Also

`hash_function_free()`

6.31.2.6 ORDO\_PUBLIC void md5\_copy ( struct MD5\_STATE \* *dst*, const struct MD5\_STATE \* *src* )

## See Also

`hash_function_copy()`

6.31.2.7 ORDO\_PUBLIC size\_t md5\_query ( int *query*, size\_t *value* )

## See Also

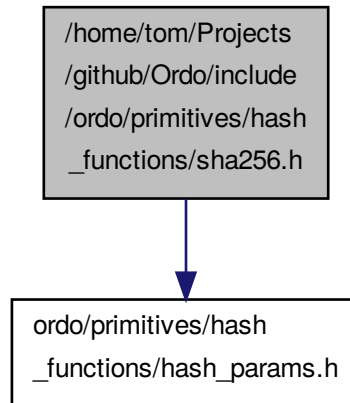
`hash_function_query()`

## 6.32 /home/tom/Projects/github/Ordo/include/ordo/primitives/hash\_functions/sha256.h File Reference

Primitive.

```
#include "ordo/primitives/hash_functions/hash_params.h"
```

Include dependency graph for sha256.h:



## Functions

- ORDO\_PUBLIC struct SHA256\_STATE \* [sha256\\_alloc](#) (void)
- ORDO\_PUBLIC int [sha256\\_init](#) (struct SHA256\_STATE \*state, const void \*params)
- ORDO\_PUBLIC void [sha256\\_update](#) (struct SHA256\_STATE \*state, const void \*buffer, size\_t len)
- ORDO\_PUBLIC void [sha256\\_final](#) (struct SHA256\_STATE \*state, void \*digest)
- ORDO\_PUBLIC void [sha256\\_free](#) (struct SHA256\_STATE \*state)
- ORDO\_PUBLIC void [sha256\\_copy](#) (struct SHA256\_STATE \*dst, const struct SHA256\_STATE \*src)
- ORDO\_PUBLIC size\_t [sha256\\_query](#) (int query, size\_t value)

### 6.32.1 Detailed Description

Primitive. The SHA-256 hash function, which produces a 256-bit digest.

### 6.32.2 Function Documentation

6.32.2.1 ORDO\_PUBLIC struct SHA256\_STATE\* [sha256\\_alloc](#) ( void )

See Also

[hash\\_function\\_alloc\(\)](#)

6.32.2.2 ORDO\_PUBLIC int [sha256\\_init](#) ( struct SHA256\_STATE \* *state*, const void \* *params* )

See Also

[hash\\_function\\_init\(\)](#)

Remarks

The `params` parameter is ignored.

6.32.2.3 ORDO\_PUBLIC void sha256\_update ( struct SHA256\_STATE \* *state*, const void \* *buffer*, size\_t *len* )

See Also

[hash\\_function\\_update\(\)](#)

6.32.2.4 ORDO\_PUBLIC void sha256\_final ( struct SHA256\_STATE \* *state*, void \* *digest* )

See Also

[hash\\_function\\_final\(\)](#)

6.32.2.5 ORDO\_PUBLIC void sha256\_free ( struct SHA256\_STATE \* *state* )

See Also

[hash\\_function\\_free\(\)](#)

6.32.2.6 ORDO\_PUBLIC void sha256\_copy ( struct SHA256\_STATE \* *dst*, const struct SHA256\_STATE \* *src* )

See Also

[hash\\_function\\_copy\(\)](#)

6.32.2.7 ORDO\_PUBLIC size\_t sha256\_query ( int *query*, size\_t *value* )

See Also

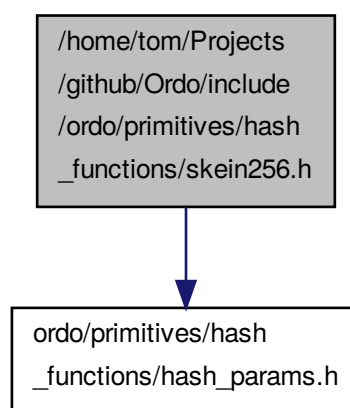
[hash\\_function\\_query\(\)](#)

## 6.33 /home/tom/Projects/github/Ordo/include/ordo/primitives/hash\_functions/skein256.h File Reference

Primitive.

```
#include "ordo/primitives/hash_functions/hash_params.h"
```

Include dependency graph for skein256.h:



### Functions

- ORDO\_PUBLIC struct SKEIN256\_STATE \* [skein256\\_alloc](#) (void)
- ORDO\_PUBLIC int [skein256\\_init](#) (struct SKEIN256\_STATE \*state, const struct [SKEIN256\\_PARAMS](#) \*params)
- ORDO\_PUBLIC void [skein256\\_update](#) (struct SKEIN256\_STATE \*state, const void \*buffer, size\_t len)
- ORDO\_PUBLIC void [skein256\\_final](#) (struct SKEIN256\_STATE \*state, void \*digest)
- ORDO\_PUBLIC void [skein256\\_free](#) (struct SKEIN256\_STATE \*state)
- ORDO\_PUBLIC void [skein256\\_copy](#) (struct SKEIN256\_STATE \*dst, const struct SKEIN256\_STATE \*src)
- ORDO\_PUBLIC size\_t [skein256\\_query](#) (int query, size\_t value)

#### 6.33.1 Detailed Description

Primitive. This is the Skein-256 hash function, which produces a 256-bit digest by default (but has parameters to output a longer digest) and has a 256-bit internal state. This implementation supports messages up to a length of  $2^{64} - 1$  bytes instead of the  $2^{96} - 1$  available, but we trust this will not be an issue. This is a rather flexible hash with lots of options. Currently, the only options supported are:

- arbitrary output length (see [SKEIN256\\_PARAMS](#))
- free access to configuration block (in fact, [SKEIN256\\_PARAMS](#) is the configuration block, and a default one is used if not provided)



## 6.33.2 Function Documentation

6.33.2.1 `ORDO_PUBLIC struct SKEIN256_STATE* skein256_alloc ( void )`

See Also

`hash_function_alloc()`

6.33.2.2 `ORDO_PUBLIC int skein256_init ( struct SKEIN256_STATE * state, const struct SKEIN256_PARAMS * params )`

See Also

`hash_function_init()`

Return values

<code>ORDO_ARG</code>	if parameters were provided, but requested an output length of zero bytes.
-----------------------	--

6.33.2.3 `ORDO_PUBLIC void skein256_update ( struct SKEIN256_STATE * state, const void * buffer, size_t len )`

See Also

`hash_function_update()`

6.33.2.4 `ORDO_PUBLIC void skein256_final ( struct SKEIN256_STATE * state, void * digest )`

See Also

`hash_function_final()`

Remarks

If no parameters are provided, the digest buffer must be at least 32 bytes (256 bits) large. If parameters are provided, the buffer must be sufficiently large to store the output length required by the parameters (note the parameters specified an output length in **bits**).

6.33.2.5 `ORDO_PUBLIC void skein256_free ( struct SKEIN256_STATE * state )`

See Also

`hash_function_free()`

6.33.2.6 `ORDO_PUBLIC void skein256_copy ( struct SKEIN256_STATE * dst, const struct SKEIN256_STATE * src )`

See Also

`hash_function_copy()`

6.33.2.7 `ORDO_PUBLIC size_t skein256_query ( int query, size_t value )`

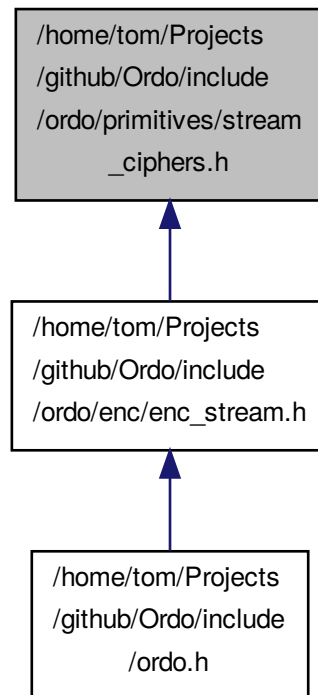
See Also

`hash_function_query()`

## 6.34 /home/tom/Projects/github/Ordo/include/ordo/primitives/stream\_ciphers.h File Reference

Abstraction Layer.

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



### Functions

- ORDO\_PUBLIC const char \* [stream\\_cipher\\_name](#) (const struct STREAM\_CIPHER \*primitive)
- ORDO\_PUBLIC const struct STREAM\_CIPHER \* [rc4](#) (void)  
*The RC4 stream cipher.*
- ORDO\_PUBLIC size\_t [stream\\_cipher\\_count](#) (void)
- ORDO\_PUBLIC const struct STREAM\_CIPHER \* [stream\\_cipher\\_by\\_name](#) (const char \*name)
- ORDO\_PUBLIC const struct STREAM\_CIPHER \* [stream\\_cipher\\_by\\_index](#) (size\_t index)
- ORDO\_PUBLIC void \* [stream\\_cipher\\_alloc](#) (const struct STREAM\_CIPHER \*primitive)
- ORDO\_PUBLIC int [stream\\_cipher\\_init](#) (const struct STREAM\_CIPHER \*primitive, void \*state, const void \*key, size\_t key\_len, const void \*params)
- ORDO\_PUBLIC void [stream\\_cipher\\_update](#) (const struct STREAM\_CIPHER \*primitive, void \*state, void \*buffer, size\_t len)
- ORDO\_PUBLIC void [stream\\_cipher\\_final](#) (const struct STREAM\_CIPHER \*primitive, void \*state)
- ORDO\_PUBLIC void [stream\\_cipher\\_free](#) (const struct STREAM\_CIPHER \*primitive, void \*state)

- ORDO\_PUBLIC void [stream\\_cipher\\_copy](#) (const struct STREAM\_CIPHER \*primitive, void \*dst, const void \*src)
- ORDO\_PUBLIC size\_t [stream\\_cipher\\_query](#) (const struct STREAM\_CIPHER \*primitive, int query, size\_t value)

### 6.34.1 Detailed Description

Abstraction Layer. This abstraction layer declares all the stream ciphers and also makes them available to higher level modules. This does not actually do encryption at all, but only abstracts stream cipher permutations, the encryption modules are in the `enc` folder: [enc\\_stream.h](#).

### 6.34.2 Function Documentation

#### 6.34.2.1 ORDO\_PUBLIC const char\* stream\_cipher\_name ( const struct STREAM\_CIPHER \* *primitive* )

Returns the name of a stream cipher primitive.

##### Parameters

<i>in</i>	<i>primitive</i>	A stream cipher primitive.
-----------	------------------	----------------------------

##### Returns

Returns the stream cipher's name.

##### Remarks

This name can then be used in [stream\\_cipher\\_by\\_name\(\)](#).

#### 6.34.2.2 ORDO\_PUBLIC size\_t stream\_cipher\_count ( void )

Exposes the number of stream ciphers available.

##### Returns

The number of available stream ciphers (at least one).

##### Remarks

This is for use in enumerating stream ciphers.

#### 6.34.2.3 ORDO\_PUBLIC const struct STREAM\_CIPHER\* stream\_cipher\_by\_name ( const char \* *name* )

Returns a stream cipher primitive from a name.

##### Parameters

<i>name</i>	A stream cipher name.
-------------	-----------------------

##### Returns

The stream cipher such that the following is true:

```
stream_cipher_name(retval) = name
```

or 0 if no such stream cipher exists.

6.34.2.4 ORDO\_PUBLIC const struct STREAM\_CIPHER\* stream\_cipher\_by\_index ( size\_t *index* )

Returns a stream cipher primitive from an index.

## Parameters

in	<i>index</i>	A stream cipher index.
----	--------------	------------------------

## Returns

The stream cipher corresponding to the provided index, or 0 if no such stream cipher exists.

## Remarks

Use `stream_cipher_count()` to obtain an upper bound on stream cipher indices (there will be at least one).

6.34.2.5 ORDO\_PUBLIC void\* stream\_cipher\_alloc ( const struct STREAM\_CIPHER \* *primitive* )

Allocates a stream cipher state.

## Parameters

in	<i>primitive</i>	A stream cipher primitive.
----	------------------	----------------------------

## Returns

An allocated stream cipher state, or 0 on error.

6.34.2.6 ORDO\_PUBLIC int stream\_cipher\_init ( const struct STREAM\_CIPHER \* *primitive*, void \* *state*, const void \* *key*, size\_t *key\_len*, const void \* *params* )

Initializes a stream cipher state.

## Parameters

in	<i>primitive</i>	A stream cipher primitive.
in, out	<i>state</i>	A stream cipher state.
in	<i>key</i>	The cryptographic key to use.
in	<i>key_len</i>	The length, in bytes, of the key.
in	<i>params</i>	Stream cipher specific parameters.

## Returns

`ORDO_SUCCESS` on success, else an error code.

6.34.2.7 ORDO\_PUBLIC void stream\_cipher\_update ( const struct STREAM\_CIPHER \* *primitive*, void \* *state*, void \* *buffer*, size\_t *len* )

Encrypts or decrypts a buffer using a stream cipher state.

## Parameters

in	<i>primitive</i>	A stream cipher primitive.
in, out	<i>state</i>	A stream cipher state.

<i>in, out</i>	<i>buffer</i>	The buffer to encrypt or decrypt.
<i>in</i>	<i>len</i>	The length, in bytes, of the buffer.

**Remarks**

Encryption and decryption are equivalent, and are done in place.

This function is stateful and will update the passed state (by generating keystream material), unlike block ciphers, which are deterministic permutations.

#### 6.34.2.8 ORDO\_PUBLIC void stream\_cipher\_final ( const struct STREAM\_CIPHER \* *primitive*, void \* *state* )

Finalizes a stream cipher state.

**Parameters**

<i>in</i>	<i>primitive</i>	A stream cipher primitive.
<i>in, out</i>	<i>state</i>	A stream cipher state.

#### 6.34.2.9 ORDO\_PUBLIC void stream\_cipher\_free ( const struct STREAM\_CIPHER \* *primitive*, void \* *state* )

Frees a stream cipher state.

**Parameters**

<i>in</i>	<i>primitive</i>	A stream cipher primitive.
<i>in, out</i>	<i>state</i>	A stream cipher state.

#### 6.34.2.10 ORDO\_PUBLIC void stream\_cipher\_copy ( const struct STREAM\_CIPHER \* *primitive*, void \* *dst*, const void \* *src* )

Copies a stream cipher state to another.

**Parameters**

<i>in</i>	<i>primitive</i>	A stream cipher primitive.
<i>out</i>	<i>dst</i>	The destination state.
<i>in</i>	<i>src</i>	The source state.

**Remarks**

The states must have been initialized with the same stream cipher and parameters, or this function's behaviour is undefined.

#### 6.34.2.11 ORDO\_PUBLIC size\_t stream\_cipher\_query ( const struct STREAM\_CIPHER \* *primitive*, int *query*, size\_t *value* )

Queries a stream cipher for suitable parameters.

**Parameters**

<i>in</i>	<i>primitive</i>	A stream cipher primitive.
<i>in</i>	<i>query</i>	A query code.
<i>in</i>	<i>value</i>	A suggested value.

**Returns**

A suitable parameter of type `query` based on `value`.

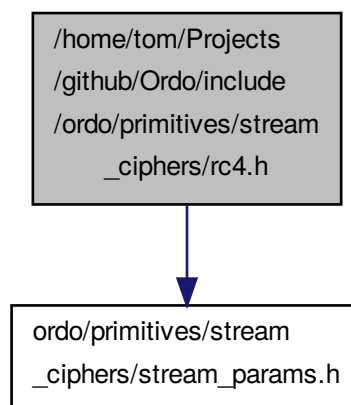
See Also

[query.h](#)

## 6.35 /home/tom/Projects/github/Ordo/include/ordo/primitives/stream\_ciphers/rc4.h File Reference

Primitive.

```
#include "ordo/primitives/stream_ciphers/stream_params.h"
Include dependency graph for rc4.h:
```



### Functions

- ORDO\_PUBLIC struct RC4\_STATE \* [rc4\\_alloc](#) (void)
- ORDO\_PUBLIC int [rc4\\_init](#) (struct RC4\_STATE \*state, const uint8\_t \*key, size\_t key\_len, const struct [RC4\\_PARAMS](#) \*params)
- ORDO\_PUBLIC void [rc4\\_update](#) (struct RC4\_STATE \*state, uint8\_t \*buffer, size\_t len)
- ORDO\_PUBLIC void [rc4\\_final](#) (struct RC4\_STATE \*state)
- ORDO\_PUBLIC void [rc4\\_free](#) (struct RC4\_STATE \*state)
- ORDO\_PUBLIC void [rc4\\_copy](#) (struct RC4\_STATE \*dst, const struct RC4\_STATE \*src)
- ORDO\_PUBLIC size\_t [rc4\\_query](#) (int query, size\_t value)

#### 6.35.1 Detailed Description

Primitive. RC4 is a stream cipher, which accepts keys between 40 and 2048 bits (in multiples of 8 bits only). It accepts a parameter consisting of the number of initial keystream bytes to drop immediately after key schedule, effectively implementing RC4-drop[n]. If no drop parameter is passed, the implementation drops 2048 bytes by default.

#### 6.35.2 Function Documentation

##### 6.35.2.1 ORDO\_PUBLIC struct RC4\_STATE\* rc4\_alloc ( void )

See Also

[stream\\_cipher\\_alloc\(\)](#)

6.35.2.2 **ORDO\_PUBLIC** int rc4\_init ( struct RC4\_STATE \* *state*, const uint8\_t \* *key*, size\_t *key\_len*, const struct RC4\_PARAMS \* *params* )

See Also

[stream\\_cipher\\_init\(\)](#)

Return values

<a href="#">ORDO_KEY_LEN</a>	if the key length was less than 40 bits (5 bytes) or more than 2048 bits (256 bytes).
------------------------------	---

Remarks

The amount of keystream bytes to drop can be set via the `params` argument, see [RC4\\_PARAMS](#). By default, 2048 bytes are dropped.

6.35.2.3 **ORDO\_PUBLIC** void rc4\_update ( struct RC4\_STATE \* *state*, uint8\_t \* *buffer*, size\_t *len* )

See Also

[stream\\_cipher\\_update\(\)](#)

6.35.2.4 **ORDO\_PUBLIC** void rc4\_final ( struct RC4\_STATE \* *state* )

See Also

[stream\\_cipher\\_final\(\)](#)

6.35.2.5 **ORDO\_PUBLIC** void rc4\_free ( struct RC4\_STATE \* *state* )

See Also

[stream\\_cipher\\_free\(\)](#)

6.35.2.6 **ORDO\_PUBLIC** void rc4\_copy ( struct RC4\_STATE \* *dst*, const struct RC4\_STATE \* *src* )

See Also

[stream\\_cipher\\_copy\(\)](#)

6.35.2.7 **ORDO\_PUBLIC** size\_t rc4\_query ( int *query*, size\_t *value* )

See Also

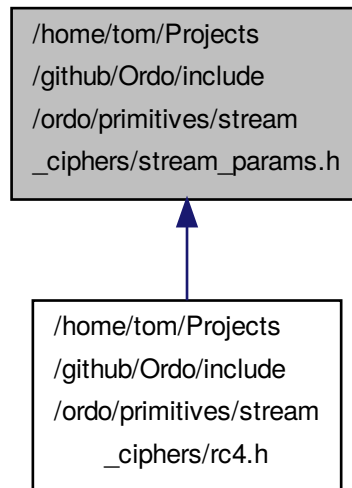
[stream\\_cipher\\_query\(\)](#)



## 6.36 /home/tom/Projects/github/Ordo/include/ordo/primitives/stream\_ciphers/stream\_params.h File Reference

Primitive Parameters.

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



### Data Structures

- struct [RC4\\_PARAMS](#)  
*RC4 stream cipher parameters.*

#### 6.36.1 Detailed Description

Primitive Parameters. This header contains parameter structures for all stream ciphers.

# Index

/home/tom/Projects/github/Ordo/include/ordo.h, [15](#)  
/home/tom/Projects/github/Ordo/include/ordo/auth/hmac.-  
h, [18](#)  
/home/tom/Projects/github/Ordo/include/ordo/common/error.-  
h, [23](#)  
/home/tom/Projects/github/Ordo/include/ordo/common/interface.-  
h, [25](#)  
/home/tom/Projects/github/Ordo/include/ordo/common/query.-  
h, [25](#)  
/home/tom/Projects/github/Ordo/include/ordo/common/version.-  
h, [27](#)  
/home/tom/Projects/github/Ordo/include/ordo/digest/digest.-  
h, [27](#)  
/home/tom/Projects/github/Ordo/include/ordo/enc/enc\_-  
block.h, [32](#)  
/home/tom/Projects/github/Ordo/include/ordo/enc/enc\_-  
stream.h, [39](#)  
/home/tom/Projects/github/Ordo/include/ordo/internal/alg.-  
h, [44](#)  
/home/tom/Projects/github/Ordo/include/ordo/internal/implementation.-  
h, [47](#)  
/home/tom/Projects/github/Ordo/include/ordo/internal/mem.-  
h, [47](#)  
/home/tom/Projects/github/Ordo/include/ordo/internal/sys.-  
h, [48](#)  
/home/tom/Projects/github/Ordo/include/ordo/kdf/pbkdf2.-  
h, [48](#)  
/home/tom/Projects/github/Ordo/include/ordo/misc/endianness.-  
h, [50](#)  
/home/tom/Projects/github/Ordo/include/ordo/misc/os\_-  
random.h, [51](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_ciphers.h, [52](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_ciphers/aes.h, [57](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_ciphers/block\_params.h, [59](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_ciphers/nullicipher.h, [60](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_ciphers/threefish256.h, [62](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_modes.h, [64](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_modes/ecb.h, [70](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_modes/cfb.h, [72](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_modes/ctr.h, [74](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_modes/ecb.h, [76](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_modes/mode\_params.h, [78](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/block-  
\_modes/ofb.h, [79](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/hash-  
\_functions.h, [81](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/hash-  
\_functions/hash\_params.h, [87](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/hash-  
\_functions/md5.h, [88](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/hash-  
\_functions/sha256.h, [89](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/hash-  
\_functions/skein256.h, [92](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/stream-  
\_ciphers.h, [94](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/stream-  
\_ciphers/rc4.h, [99](#)  
/home/tom/Projects/github/Ordo/include/ordo/primitives/stream-  
\_ciphers/stream\_params.h, [101](#)  
AES\_PARAMS, [9](#)  
    rounds, [9](#)  
aes.h  
    aes\_alloc, [58](#)  
    aes\_copy, [59](#)  
    aes\_final, [58](#)  
    aes\_forward, [58](#)  
    aes\_free, [59](#)  
    aes\_init, [58](#)  
    aes\_inverse, [58](#)  
    aes\_query, [59](#)  
aes\_alloc  
aes.h, [58](#)  
aes\_copy  
aes.h, [59](#)  
aes\_final  
aes.h, [58](#)  
aes\_forward  
aes.h, [58](#)  
aes\_free  
aes.h, [59](#)  
aes\_init  
aes.h, [58](#)  
aes\_inverse  
aes.h, [58](#)  
aes\_query  
aes.h, [59](#)

- alg.h
  - bits, [44](#)
  - bytes, [44](#)
  - inc\_buffer, [46](#)
  - offset, [44](#)
  - pad\_check, [46](#)
  - xor\_buffer, [46](#)
- BLOCK\_SIZE\_Q
  - query.h, [26](#)
- bits
  - alg.h, [44](#)
- block\_cipher\_alloc
  - block\_ciphers.h, [54](#)
- block\_cipher\_by\_index
  - block\_ciphers.h, [54](#)
- block\_cipher\_by\_name
  - block\_ciphers.h, [53](#)
- block\_cipher\_copy
  - block\_ciphers.h, [56](#)
- block\_cipher\_count
  - block\_ciphers.h, [53](#)
- block\_cipher\_final
  - block\_ciphers.h, [56](#)
- block\_cipher\_forward
  - block\_ciphers.h, [54](#)
- block\_cipher\_free
  - block\_ciphers.h, [56](#)
- block\_cipher\_init
  - block\_ciphers.h, [54](#)
- block\_cipher\_inverse
  - block\_ciphers.h, [56](#)
- block\_cipher\_name
  - block\_ciphers.h, [53](#)
- block\_cipher\_query
  - block\_ciphers.h, [57](#)
- block\_ciphers.h
  - block\_cipher\_alloc, [54](#)
  - block\_cipher\_by\_index, [54](#)
  - block\_cipher\_by\_name, [53](#)
  - block\_cipher\_copy, [56](#)
  - block\_cipher\_count, [53](#)
  - block\_cipher\_final, [56](#)
  - block\_cipher\_forward, [54](#)
  - block\_cipher\_free, [56](#)
  - block\_cipher\_init, [54](#)
  - block\_cipher\_inverse, [56](#)
  - block\_cipher\_name, [53](#)
  - block\_cipher\_query, [57](#)
- block\_mode\_alloc
  - block\_modes.h, [67](#)
- block\_mode\_by\_index
  - block\_modes.h, [67](#)
- block\_mode\_by\_name
  - block\_modes.h, [66](#)
- block\_mode\_copy
  - block\_modes.h, [70](#)
- block\_mode\_count
  - block\_modes.h, [66](#)
- block\_mode\_final
  - block\_modes.h, [68](#)
- block\_mode\_free
  - block\_modes.h, [68](#)
- block\_mode\_init
  - block\_modes.h, [67](#)
- block\_mode\_name
  - block\_modes.h, [66](#)
- block\_mode\_query
  - block\_modes.h, [70](#)
- block\_mode\_update
  - block\_modes.h, [68](#)
- block\_modes.h
  - block\_mode\_alloc, [67](#)
  - block\_mode\_by\_index, [67](#)
  - block\_mode\_by\_name, [66](#)
  - block\_mode\_copy, [70](#)
  - block\_mode\_count, [66](#)
  - block\_mode\_final, [68](#)
  - block\_mode\_free, [68](#)
  - block\_mode\_init, [67](#)
  - block\_mode\_name, [66](#)
  - block\_mode\_query, [70](#)
  - block\_mode\_update, [68](#)
- bytes
  - alg.h, [44](#)
- CBC\_PARAMS, [9](#)
  - padding, [10](#)
- cbc.h
  - cbc\_alloc, [71](#)
  - cbc\_copy, [72](#)
  - cbc\_final, [72](#)
  - cbc\_free, [72](#)
  - cbc\_init, [72](#)
  - cbc\_query, [72](#)
  - cbc\_update, [72](#)
- cbc\_alloc
  - cbc.h, [71](#)
- cbc\_copy
  - cbc.h, [72](#)
- cbc\_final
  - cbc.h, [72](#)
- cbc\_free
  - cbc.h, [72](#)
- cbc\_init
  - cbc.h, [72](#)
- cbc\_query
  - cbc.h, [72](#)
- cbc\_update
  - cbc.h, [72](#)
- cfb.h
  - cfb\_alloc, [73](#)
  - cfb\_copy, [74](#)
  - cfb\_final, [74](#)
  - cfb\_free, [74](#)
  - cfb\_init, [74](#)
  - cfb\_query, [74](#)
  - cfb\_update, [74](#)

- cfb\_alloc
  - cfb.h, 73
- cfb\_copy
  - cfb.h, 74
- cfb\_final
  - cfb.h, 74
- cfb\_free
  - cfb.h, 74
- cfb\_init
  - cfb.h, 74
- cfb\_query
  - cfb.h, 74
- cfb\_update
  - cfb.h, 74
- ctr.h
  - ctr\_alloc, 75
  - ctr\_copy, 76
  - ctr\_final, 76
  - ctr\_free, 76
  - ctr\_init, 76
  - ctr\_query, 76
  - ctr\_update, 76
- ctr\_alloc
  - ctr.h, 75
- ctr\_copy
  - ctr.h, 76
- ctr\_final
  - ctr.h, 76
- ctr\_free
  - ctr.h, 76
- ctr\_init
  - ctr.h, 76
- ctr\_query
  - ctr.h, 76
- ctr\_update
  - ctr.h, 76
- DIGEST\_LEN\_Q
  - query.h, 26
- digest.h
  - digest\_alloc, 30
  - digest\_copy, 31
  - digest\_final, 31
  - digest\_free, 31
  - digest\_init, 30
  - digest\_length, 32
  - digest\_update, 30
- digest\_alloc
  - digest.h, 30
- digest\_copy
  - digest.h, 31
- digest\_final
  - digest.h, 31
- digest\_free
  - digest.h, 31
- digest\_init
  - digest.h, 30
- digest\_length
  - digest.h, 32
- digest\_update
  - digest.h, 30
- drop
  - RC4\_PARAMS, 11
- ECB\_PARAMS, 10
  - padding, 10
- ecb.h
  - ecb\_alloc, 77
  - ecb\_copy, 78
  - ecb\_final, 78
  - ecb\_free, 78
  - ecb\_init, 78
  - ecb\_query, 78
  - ecb\_update, 78
- ecb\_alloc
  - ecb.h, 77
- ecb\_copy
  - ecb.h, 78
- ecb\_final
  - ecb.h, 78
- ecb\_free
  - ecb.h, 78
- ecb\_init
  - ecb.h, 78
- ecb\_query
  - ecb.h, 78
- ecb\_update
  - ecb.h, 78
- enc\_block.h
  - enc\_block\_alloc, 33
  - enc\_block\_copy, 37
  - enc\_block\_final, 35
  - enc\_block\_free, 37
  - enc\_block\_init, 35
  - enc\_block\_iv\_len, 37
  - enc\_block\_key\_len, 37
  - enc\_block\_update, 35
- enc\_block\_alloc
  - enc\_block.h, 33
- enc\_block\_copy
  - enc\_block.h, 37
- enc\_block\_final
  - enc\_block.h, 35
- enc\_block\_free
  - enc\_block.h, 37
- enc\_block\_init
  - enc\_block.h, 35
- enc\_block\_iv\_len
  - enc\_block.h, 37
- enc\_block\_key\_len
  - enc\_block.h, 37
- enc\_block\_update
  - enc\_block.h, 35
- enc\_stream.h
  - enc\_stream\_alloc, 40
  - enc\_stream\_copy, 42
  - enc\_stream\_final, 42
  - enc\_stream\_free, 42

- enc\_stream\_init, [40](#)
  - enc\_stream\_key\_len, [42](#)
  - enc\_stream\_update, [42](#)
- enc\_stream\_alloc
  - enc\_stream.h, [40](#)
- enc\_stream\_copy
  - enc\_stream.h, [42](#)
- enc\_stream\_final
  - enc\_stream.h, [42](#)
- enc\_stream\_free
  - enc\_stream.h, [42](#)
- enc\_stream\_init
  - enc\_stream.h, [40](#)
- enc\_stream\_key\_len
  - enc\_stream.h, [42](#)
- enc\_stream\_update
  - enc\_stream.h, [42](#)
- error.h
  - ORDO\_ALLOC, [24](#)
  - ORDO\_ARG, [24](#)
  - ORDO\_FAIL, [23](#)
  - ORDO\_KEY\_LEN, [24](#)
  - ORDO\_LEFTOVER, [24](#)
  - ORDO\_PADDING, [24](#)
  - ORDO\_SUCCESS, [23](#)
- error.h
  - ORDO\_ERROR, [23](#)
  - ordo\_error\_msg, [24](#)
- hash\_function\_alloc
  - hash\_functions.h, [85](#)
- hash\_function\_by\_index
  - hash\_functions.h, [85](#)
- hash\_function\_by\_name
  - hash\_functions.h, [83](#)
- hash\_function\_copy
  - hash\_functions.h, [86](#)
- hash\_function\_count
  - hash\_functions.h, [83](#)
- hash\_function\_final
  - hash\_functions.h, [86](#)
- hash\_function\_free
  - hash\_functions.h, [86](#)
- hash\_function\_init
  - hash\_functions.h, [85](#)
- hash\_function\_name
  - hash\_functions.h, [83](#)
- hash\_function\_query
  - hash\_functions.h, [86](#)
- hash\_function\_update
  - hash\_functions.h, [85](#)
- hash\_functions.h
  - hash\_function\_alloc, [85](#)
  - hash\_function\_by\_index, [85](#)
  - hash\_function\_by\_name, [83](#)
  - hash\_function\_copy, [86](#)
  - hash\_function\_count, [83](#)
  - hash\_function\_final, [86](#)
  - hash\_function\_free, [86](#)
  - hash\_function\_init, [85](#)
  - hash\_function\_name, [83](#)
  - hash\_function\_query, [86](#)
  - hash\_function\_update, [85](#)
- hash\_function\_init, [85](#)
- hash\_function\_name, [83](#)
- hash\_function\_query, [86](#)
- hash\_function\_update, [85](#)
- hmac.h
  - hmac\_alloc, [20](#)
  - hmac\_copy, [22](#)
  - hmac\_final, [22](#)
  - hmac\_free, [22](#)
  - hmac\_init, [20](#)
  - hmac\_update, [20](#)
- hmac\_alloc
  - hmac.h, [20](#)
- hmac\_copy
  - hmac.h, [22](#)
- hmac\_final
  - hmac.h, [22](#)
- hmac\_free
  - hmac.h, [22](#)
- hmac\_init
  - hmac.h, [20](#)
- hmac\_update
  - hmac.h, [20](#)
- IV\_LEN\_Q
  - query.h, [27](#)
- inc\_buffer
  - alg.h, [46](#)
- KEY\_LEN\_Q
  - query.h, [26](#)
- md5.h
  - md5\_alloc, [88](#)
  - md5\_copy, [89](#)
  - md5\_final, [89](#)
  - md5\_free, [89](#)
  - md5\_init, [88](#)
  - md5\_query, [89](#)
  - md5\_update, [89](#)
- md5\_alloc
  - md5.h, [88](#)
- md5\_copy
  - md5.h, [89](#)
- md5\_final
  - md5.h, [89](#)
- md5\_free
  - md5.h, [89](#)
- md5\_init
  - md5.h, [88](#)
- md5\_query
  - md5.h, [89](#)
- md5\_update
  - md5.h, [89](#)
- mem.h
  - mem\_alloc, [47](#)
  - mem\_erase, [48](#)
  - mem\_free, [48](#)
- mem\_alloc

- mem.h, 47
- mem\_erase
  - mem.h, 48
- mem\_free
  - mem.h, 48
- nullcipher.h
  - nullcipher\_alloc, 61
  - nullcipher\_copy, 61
  - nullcipher\_final, 61
  - nullcipher\_forward, 61
  - nullcipher\_free, 61
  - nullcipher\_init, 61
  - nullcipher\_inverse, 61
  - nullcipher\_query, 61
- nullcipher\_alloc
  - nullcipher.h, 61
- nullcipher\_copy
  - nullcipher.h, 61
- nullcipher\_final
  - nullcipher.h, 61
- nullcipher\_forward
  - nullcipher.h, 61
- nullcipher\_free
  - nullcipher.h, 61
- nullcipher\_init
  - nullcipher.h, 61
- nullcipher\_inverse
  - nullcipher.h, 61
- nullcipher\_query
  - nullcipher.h, 61
- ORDO\_ALLOC
  - error.h, 24
- ORDO\_ARG
  - error.h, 24
- ORDO\_FAIL
  - error.h, 23
- ORDO\_KEY\_LEN
  - error.h, 24
- ORDO\_LEFTOVER
  - error.h, 24
- ORDO\_PADDING
  - error.h, 24
- ORDO\_SUCCESS
  - error.h, 23
- ORDO\_ERROR
  - error.h, 23
- ORDO\_QUERY
  - query.h, 26
- ORDO\_VERSION, 10
- ofb.h
  - ofb\_alloc, 80
  - ofb\_copy, 81
  - ofb\_final, 80
  - ofb\_free, 80
  - ofb\_init, 80
  - ofb\_query, 81
  - ofb\_update, 80
- ofb\_alloc
  - ofb.h, 80
- ofb\_copy
  - ofb.h, 81
- ofb\_final
  - ofb.h, 80
- ofb\_free
  - ofb.h, 80
- ofb\_init
  - ofb.h, 80
- ofb\_query
  - ofb.h, 81
- ofb\_update
  - ofb.h, 80
- offset
  - alg.h, 44
- ordo.h
  - ordo\_allocator, 16
  - ordo\_digest, 17
  - ordo\_enc\_block, 16
  - ordo\_enc\_stream, 17
  - ordo\_hmac, 18
- ordo\_allocator
  - ordo.h, 16
- ordo\_digest
  - ordo.h, 17
- ordo\_enc\_block
  - ordo.h, 16
- ordo\_enc\_stream
  - ordo.h, 17
- ordo\_error\_msg
  - error.h, 24
- ordo\_hmac
  - ordo.h, 18
- os\_random
  - os\_random.h, 51
- os\_random.h
  - os\_random, 51
- out\_len
  - SKEIN256\_PARAMS, 12
- pad\_check
  - alg.h, 46
- padding
  - CBC\_PARAMS, 10
  - ECB\_PARAMS, 10
- pbkdf2
  - pbkdf2.h, 50
- pbkdf2.h
  - pbkdf2, 50
- query.h
  - BLOCK\_SIZE\_Q, 26
  - DIGEST\_LEN\_Q, 26
  - IV\_LEN\_Q, 27
  - KEY\_LEN\_Q, 26
- query.h
  - ORDO\_QUERY, 26

- RC4\_PARAMS, 11
  - drop, 11
- rc4.h
  - rc4\_alloc, 99
  - rc4\_copy, 100
  - rc4\_final, 100
  - rc4\_free, 100
  - rc4\_init, 100
  - rc4\_query, 100
  - rc4\_update, 100
- rc4\_alloc
  - rc4.h, 99
- rc4\_copy
  - rc4.h, 100
- rc4\_final
  - rc4.h, 100
- rc4\_free
  - rc4.h, 100
- rc4\_init
  - rc4.h, 100
- rc4\_query
  - rc4.h, 100
- rc4\_update
  - rc4.h, 100
- rounds
  - AES\_PARAMS, 9
- SKEIN256\_PARAMS, 12
  - out\_len, 12
- sha256.h
  - sha256\_alloc, 90
  - sha256\_copy, 91
  - sha256\_final, 91
  - sha256\_free, 91
  - sha256\_init, 90
  - sha256\_query, 91
  - sha256\_update, 90
- sha256\_alloc
  - sha256.h, 90
- sha256\_copy
  - sha256.h, 91
- sha256\_final
  - sha256.h, 91
- sha256\_free
  - sha256.h, 91
- sha256\_init
  - sha256.h, 90
- sha256\_query
  - sha256.h, 91
- sha256\_update
  - sha256.h, 90
- skein256.h
  - skein256\_alloc, 93
  - skein256\_copy, 93
  - skein256\_final, 93
  - skein256\_free, 93
  - skein256\_init, 93
  - skein256\_query, 93
  - skein256\_update, 93
- skein256\_alloc
  - skein256.h, 93
- skein256\_copy
  - skein256.h, 93
- skein256\_final
  - skein256.h, 93
- skein256\_free
  - skein256.h, 93
- skein256\_init
  - skein256.h, 93
- skein256\_query
  - skein256.h, 93
- skein256\_update
  - skein256.h, 93
- stream\_cipher\_alloc
  - stream\_ciphers.h, 97
- stream\_cipher\_by\_index
  - stream\_ciphers.h, 95
- stream\_cipher\_by\_name
  - stream\_ciphers.h, 95
- stream\_cipher\_copy
  - stream\_ciphers.h, 98
- stream\_cipher\_count
  - stream\_ciphers.h, 95
- stream\_cipher\_final
  - stream\_ciphers.h, 98
- stream\_cipher\_free
  - stream\_ciphers.h, 98
- stream\_cipher\_init
  - stream\_ciphers.h, 97
- stream\_cipher\_name
  - stream\_ciphers.h, 95
- stream\_cipher\_query
  - stream\_ciphers.h, 98
- stream\_cipher\_update
  - stream\_ciphers.h, 97
- stream\_ciphers.h
  - stream\_cipher\_alloc, 97
  - stream\_cipher\_by\_index, 95
  - stream\_cipher\_by\_name, 95
  - stream\_cipher\_copy, 98
  - stream\_cipher\_count, 95
  - stream\_cipher\_final, 98
  - stream\_cipher\_free, 98
  - stream\_cipher\_init, 97
  - stream\_cipher\_name, 95
  - stream\_cipher\_query, 98
  - stream\_cipher\_update, 97
- THREEFISH256\_PARAMS, 13
- threefish256.h
  - threefish256\_alloc, 63
  - threefish256\_copy, 63
  - threefish256\_final, 63
  - threefish256\_forward, 63
  - threefish256\_free, 63
  - threefish256\_init, 63
  - threefish256\_inverse, 63
  - threefish256\_query, 63

threefish256\_alloc  
    threefish256.h, [63](#)  
threefish256\_copy  
    threefish256.h, [63](#)  
threefish256\_final  
    threefish256.h, [63](#)  
threefish256\_forward  
    threefish256.h, [63](#)  
threefish256\_free  
    threefish256.h, [63](#)  
threefish256\_init  
    threefish256.h, [63](#)  
threefish256\_inverse  
    threefish256.h, [63](#)  
threefish256\_query  
    threefish256.h, [63](#)  
  
xor\_buffer  
    alg.h, [46](#)