Ordo

2.7.1

Generated by Doxygen 1.8.6

Wed Apr 2 2014 15:33:15

Contents

Chapter 1

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 2

Ordo v2.7.1

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 2.7.1:

- the test driver is being reworked (work in progress)
- internal functions have been namespaced, so they will no longer cause name conflicts when linking statically
- primitive functions like `rc4()` have been prefixed with `ordo_` to prevent name conflicts (e.g. `ordo_ctr()`)
- added `os_secure_random()` function, see the documentation for more information

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	Authentication	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 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

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:

- /ssd/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:

- /ssd/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:

- /ssd/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:

- /ssd/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:

- /ssd/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:

- /ssd/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:

- /ssd/Ordo/include/ordo/primitives/block_ciphers/block_params.h

Chapter 6

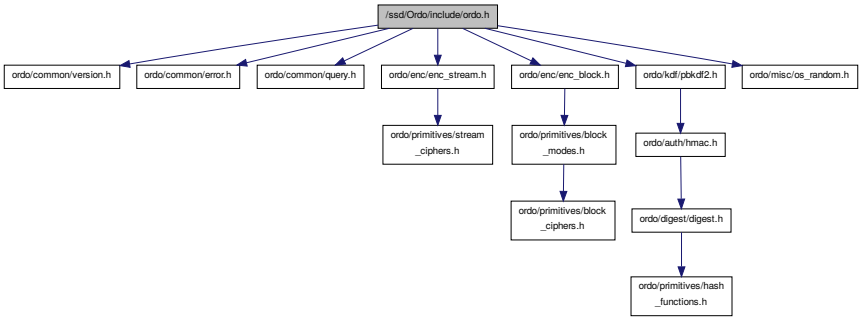
File Documentation

6.1 /ssd/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	*alloc*	The allocation function.
in	*free*	The deallocation function.
in	*data*	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	*cipher*	The block cipher to use.
in	*cipher_params*	The block cipher parameters.
in	*mode*	The mode of operation to use.
in	*mode_params*	The mode of operation parameters.

in	*direction*	1 for encryption, 0 for decryption.
in	*key*	The cryptographic key to use.
in	*key_len*	The length in bytes of the key.
in	*iv*	The initialization vector.
in	*iv_len*	The length in bytes of the IV.
in	*in*	The input plaintext/ciphertext buffer.
in	*in_len*	The length of the input buffer.
out	*out*	The output ciphertext/plaintext buffer.
out	*out_len*	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	*cipher*	The stream cipher to use.
in	*params*	The stream cipher parameters.
in,out	*inout*	The plaintext or ciphertext buffer.
in	*len*	The length, in bytes, of the buffer.
in	*key*	The cryptographic key to use.
in	*key_len*	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	*hash*	The hash function to use.
in	*params*	The hash function parameters.
in	*in*	The input buffer to hash.
in	*in_len*	The length in bytes of the buffer.
out	*digest*	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	*hash*	The hash function to use.
in	*params*	The hash function parameters.
in	*key*	The key to use for authentication.
in	*key_len*	The length in bytes of the key.
in	*in*	The input buffer to authenticate.
in	*in_len*	The length, in bytes, of the input buffer.
out	*fingerprint*	The output buffer for the fingerprint.

Returns

ORDO_SUCCESS on success, else an error code.

Remarks

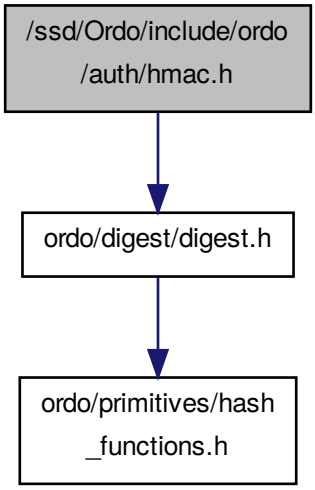
Do not use hash parameters which modify output length.

6.2 /ssd/Ordo/include/ordo/auth/hmac.h File Reference

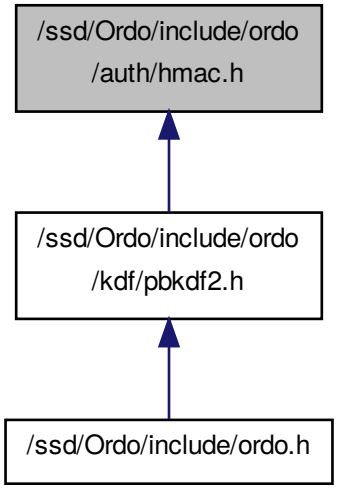
Module.

```
#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

in	*hash*	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

in	*ctx*	An allocated HMAC context.
in	*key*	The cryptographic key to use.
in	*key_len*	The size, in bytes, of the key.
out	*params*	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	*ctx*	An initialized HMAC context.
in	*in*	The data to feed into the context.
in	*in_len*	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	*ctx*	An initialized HMAC context.
out	*fingerprint*	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	*ctx*	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	*dst*	The destination context.
in	*src*	The source context.

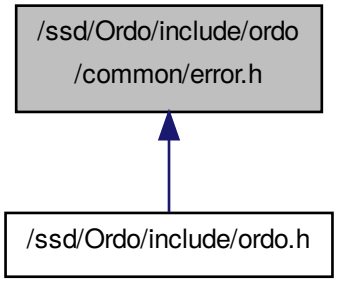
Remarks

The destination context should have been allocated using the same primitive(s) as the source context, and mustn't be initialized.
The source context must be initialized.

6.3 /ssd/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.
The `block_cipher_query()` function can be used to select a suitable key length for a given block cipher via the `#KEY_LEN` query code. For stream ciphers, use `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 `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 **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.
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 overriden) 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.
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

in	*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 /ssd/Ordo/include/ordo/common/interface.h File Reference

API.

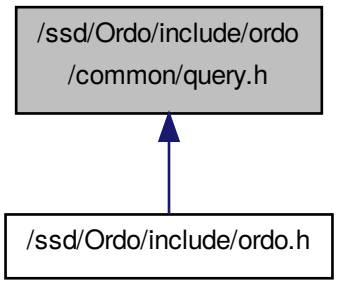
6.4.1 Detailed Description

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 /ssd/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

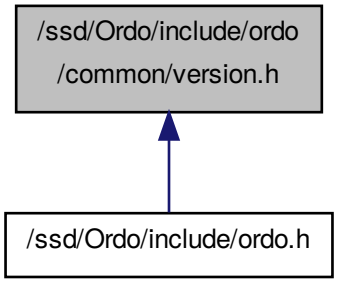
Remarks

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 /ssd/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

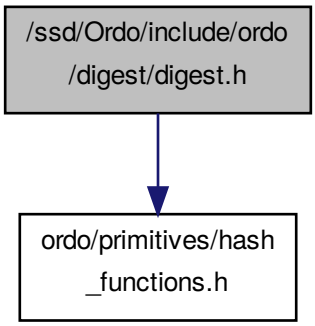
Utility. This header exposes functionality relating to the library's version.

6.7 /ssd/Ordo/include/ordo/digest/digest.h File Reference

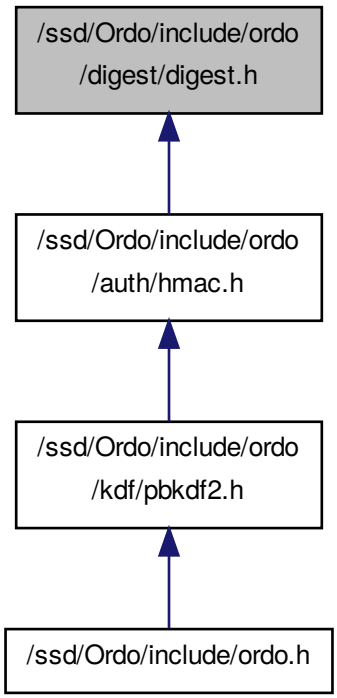
Module.

```
#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`	*hash*	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`	*ctx*	An allocated digest context.

in	*params*	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

in,out	*ctx*	An initialized digest context.
in	*in*	The data to feed into the context.
in	*in_len*	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

in,out	*ctx*	An initialized digest context.
out	*digest*	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

in	*ctx*	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	*dst*	The destination context.
in	*src*	The source context.

Remarks

The destination context should have been allocated using the same primitive(s) as the source context, and mustn't be initialized.
The source context must be initialized.
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	*hash*	A hash function primitive.

Returns

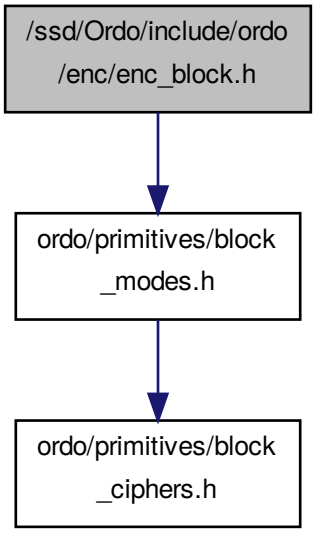
The length of the digest to be written in the `digest` parameter of `digest_final()`, if no parameters which affect output length were provided to `digest_init()`.

6.8 /ssd/Ordo/include/ordo/enc/enc_block.h File Reference

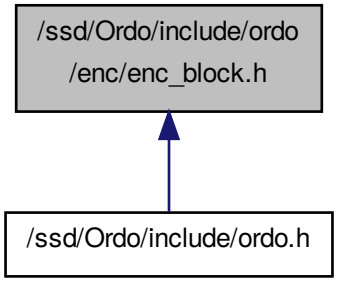
Module.

```
#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`	*cipher*	The block cipher to use.
`in`	*mode*	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`	*ctx*	A block encryption context.
`in`	*key*	The cryptographic key to use.

in	*key_len*	The length, in bytes, of the key.
in	*iv*	The initialization vector to use.
in	*iv_len*	The length, in bytes, of the IV.
in	*direction*	1 for encryption, 0 for decryption.
in	*cipher_params*	Block cipher specific parameters.
in	*mode_params*	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	*ctx*	A block encryption context.
in	*in*	The plaintext or ciphertext buffer.
in	*in_len*	Length, in bytes, of the input buffer.
out	*out*	The ciphertext or plaintext buffer.
out	*out_len*	The number of bytes written to out.

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

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

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

`in,out`	*ctx*	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

`out`	*dst*	The destination context.
`in`	*src*	The source context.

Remarks

The destination context should have been allocated using the same primitive(s) as the source context, and mustn't be initialized.
The source context must be initialized.

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

`in`	*cipher*	A block cipher primitive.
`in`	*key_len*	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	*cipher*	A block cipher primitive.
in	*mode*	A block mode primitive.
in	*iv_len*	A suggested IV length.

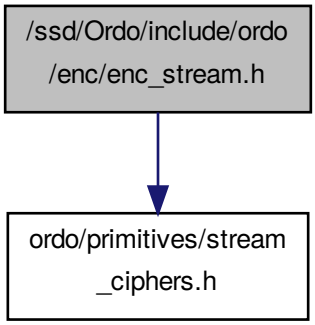
Returns

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

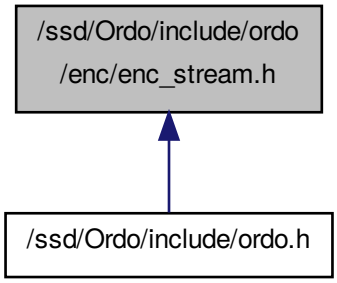
6.9 /ssd/Ordo/include/ordo/enc/enc_stream.h File Reference

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

in	*cipher*	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	*ctx*	A stream encryption context.
in	*key*	The cryptographic key to use.
in	*key_size*	The size, in bytes, of the key.
in	*params*	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	*ctx*	A stream encryption context.
in,out	*buffer*	The plaintext or ciphertext buffer.

in	*len*	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	*ctx*	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	*ctx*	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	*dst*	The destination context.
in	*src*	The source context.

Remarks

The destination context should have been allocated using the same primitive(s) as the source context, and mustn't be initialized.
The source context must be initialized.

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	*cipher*	The stream cipher to probe.
in	*key_len*	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 /ssd/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.

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`	*ptr*	Base pointer.
`in`	*len*	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	*buffer*	The buffer to check, starting at the first padding byte.
in	*padding*	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	*dst*	The destination buffer.
in	*src*	The source buffer.
in	*len*	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

in,out	*buffer*	The buffer to increment in-place.
in	*len*	The size, in bytes, of the buffer.

Remarks

Carry propagation is done left-to-right.

6.11 /ssd/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 /ssd/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

in	*size*	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

in	*ptr*	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

in,out	*ptr*	The memory buffer to overwrite.
in	*size*	The number of bytes to overwrite.

6.13 /ssd/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.

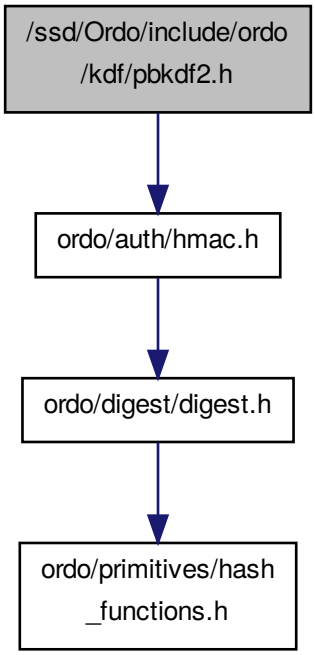
See alg.h about internal headers.

6.14 /ssd/Ordo/include/ordo/kdf/pbkdf2.h File Reference

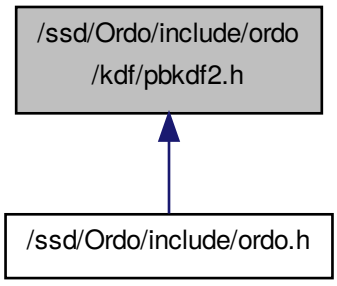
Module.

```
#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	*hash*	The hash function to use (the PRF used will be an instantiation of HMAC with it)
in	*params*	Hash-specific parameters.
in	*password*	The password to derive a key from.
in	*password_len*	The length in bytes of the password.
in	*salt*	The cryptographic salt to use.
in	*salt_len*	The length in bytes of the salt.
in	*iterations*	The number of PBKDF2 iterations to use.
out	*out*	The output buffer for the derived key.
in	*out_len*	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 /ssd/Ordo/include/ordo/misc/endianness.h File Reference

Utility.

6.15.1 Detailed Description

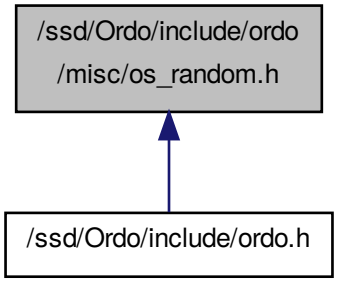
Utility. This header provides endianness functionality. You may use it freely as it has a stable API and is public. Only supports little/big endian for now.

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

6.16 /ssd/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)
- ORDO_PUBLIC int os_secure_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. To learn more about how it is implemented, go to the source code or find out what facilities your operating system provides for entropy gathering.

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.
in	*len*	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.

6.16.2.2 ORDO_PUBLIC int os_secure_random (void * *out*, size_t *len*)

Generates cryptographically secure pseudorandom numbers, the function will make a best effort attempt to access the operating system entropy pool and so, ideally, should return exactly len bytes of entropy, whereas the os-

`_random` function need only return *enough* entropy for the output stream to be computationally indistinguishable from a non-random stream. However, keep in mind that this function is **not required** to behave as such.

Parameters

out	*out*	The destination buffer.
in	*len*	The number of bytes to generate.

Returns

ORDO_SUCCESS on success, else an error code.

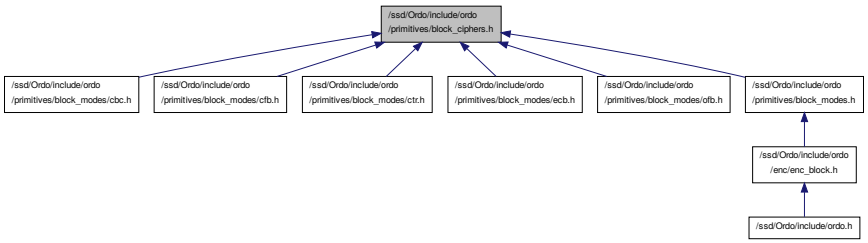
Remarks

If your platform doesn't provide this feature, this function will fall back to os_random() (there is no way to know whether this feature is available, this is by design).
You should not need to know whether this feature is available, as this function will make a "best effort" attempt to obtain entropy from the operating system - you should use this function for high security uses such as generating private keys (it has a high cost so don't use it for e.g. nonces and initialization vectors).

6.17 /ssd/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 * ordo_nullcipher (void)

  The NullCipher block cipher.
- ORDO_PUBLIC const struct BLOCK_CIPHER * ordo_threefish256 (void)

  The Threefish-256 block cipher.
- ORDO_PUBLIC const struct BLOCK_CIPHER * ordo_aes (void)

  The AES block cipher.
- 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 size_t block_cipher_count (void)
- 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

`in`	*primitive*	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 const struct BLOCK_CIPHER* block_cipher_by_name (const char * *name*)

Returns a block cipher primitive from a name.

Parameters

name	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.

6.17.2.3 ORDO_PUBLIC const struct BLOCK_CIPHER* block_cipher_by_index (size_t *index*)

Returns a block cipher primitive from an index.

Parameters

in	*index*	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).

6.17.2.4 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.

6.17.2.5 ORDO_PUBLIC void* block_cipher_alloc (const struct BLOCK_CIPHER * *primitive*)

Allocates a block cipher state.

Parameters

in	*primitive*	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

in	*primitive*	A block cipher primitive.
in,out	*state*	An allocated block cipher state.
in	*key*	The cryptographic key to use.
in	*key_len*	The length, in bytes, of the key.
in	*params*	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	*primitive*	A block cipher primitive.
in	*state*	An initialized block cipher state.
in,out	*block*	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	*primitive*	A block cipher primitive.
in	*state*	An initialized block cipher state.
in,out	*block*	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	*primitive*	A block cipher primitive.
in,out	*state*	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	*primitive*	A block cipher primitive.
in,out	*state*	A block cipher state.

6.17.2.11 ORDO_PUBLIC void block_cipher_copy (const struct BLOCK_CIPHER * *primitive,* void * *dst,* const void * *src*)

Performs a deep copy of one state into another.

Parameters

in	*primitive*	A block cipher primitive.
out	*dst*	The destination state.

in	*src*	The source state.

Remarks

The destination state must have been allocated, by using the same primitive(s) as the source state, and mustn't be initialized.
The source state must be initialized.

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

in	*primitive*	A block cipher primitive.
in	*query*	A query code.
in	*value*	A suggested value.

Returns

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

See Also

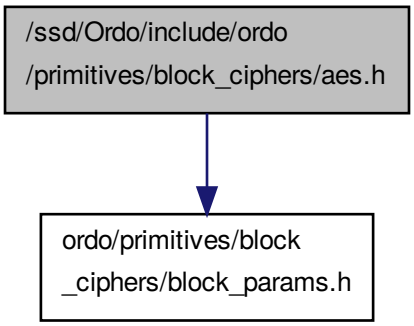
query.h

6.18 /ssd/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

ORDO_KEY_LEN	if the key length is not 16, 24, or 32 (bytes).
ORDO_ARG	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*)

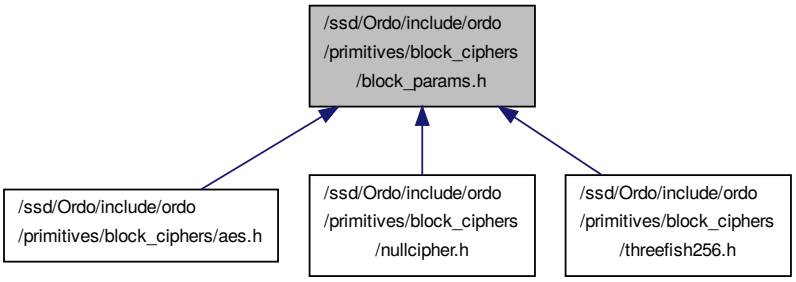
See Also

`block_cipher_query()`

6.19 /ssd/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

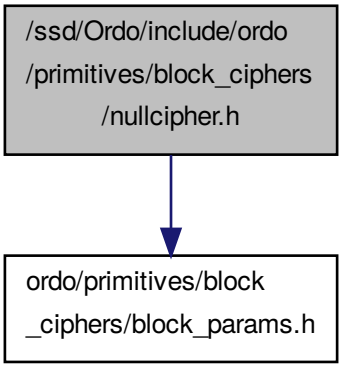
Primitive Parameters. This header contains parameter structures for all block ciphers.

6.20 /ssd/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

ORDO_KEY_LEN	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

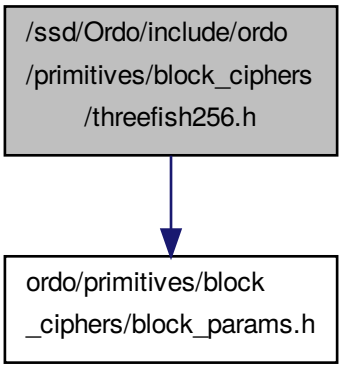
block_cipher_query()

6.21 /ssd/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

ORDO_KEY_LEN	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

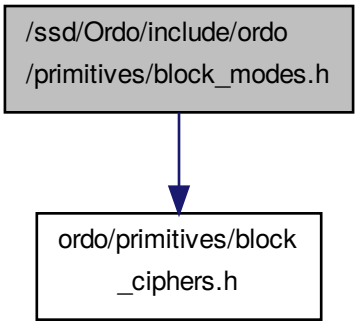
block_cipher_query()

6.22 /ssd/Ordo/include/ordo/primitives/block_modes.h File Reference

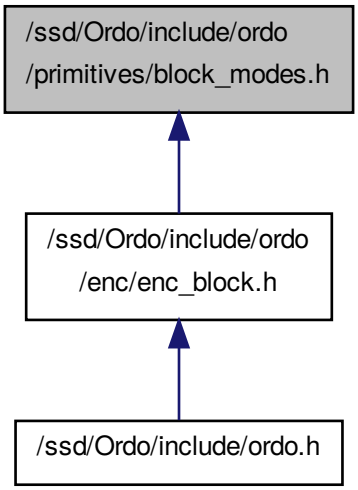
Abstraction Layer.

```
#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 * ordo_ecb (void)

  The ECB (Electronic CodeBook) block mode of operation.
- ORDO_PUBLIC const struct BLOCK_MODE * ordo_cbc (void)

  The CBC (Ciphertext Block Chaining) block mode of operation.

- ORDO_PUBLIC const struct BLOCK_MODE ∗ ordo_ctr (void)

  The CTR (CounTeR) block mode of operation.
- ORDO_PUBLIC const struct BLOCK_MODE ∗ ordo_cfb (void)

  The CFB (Cipher FeedBack) block mode of operation.
- ORDO_PUBLIC const struct BLOCK_MODE ∗ ordo_ofb (void)

  The OFB (Output FeedBack) block mode of operation.
- 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 size_t block_mode_count (void)
- 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	*mode*	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 const struct BLOCK_MODE* block_mode_by_name (const char * *name*)

Returns a block mode primitive from a name.

Parameters

name	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.

6.22.2.3 ORDO_PUBLIC const struct BLOCK_MODE* block_mode_by_index (size_t *index*)

Returns a block cipher mode from an index.

Parameters

in	*index*	A block mode index.

Returns

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

6.22.2.4 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.

6.22.2.5 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

in	*mode*	A block mode primitive.
in	*cipher*	A block cipher primitive.
in	*cipher_state*	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

in	*mode*	A block mode primitive.
in,out	*state*	A block mode state.
in	*cipher*	A block cipher primitive.
in	*cipher_state*	A block cipher state.
in	*iv*	The initialization vector to use.
in	*iv_len*	The length, in bytes, of the IV.
in	*direction*	1 for encryption, 0 for decryption.
in	*params*	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	*mode*	A block mode primitive.
in,out	*state*	A block mode state.
in	*cipher*	A block cipher primitive.
in	*cipher_state*	A block cipher state.
in	*in*	The input buffer.
in	*in_len*	The length, in bytes, of the input.
out	*out*	The output buffer.
out	*out_len*	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 **not** 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	*mode*	A block mode primitive.
in,out	*state*	A block mode state.
in	*cipher*	A block cipher primitive.
in	*cipher_state*	A block cipher state.
out	*out*	The output buffer.
out	*out_len*	A pointer to an integer to which to store the number of bytes written to out.

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	*mode*	A block mode primitive.
in,out	*state*	A block mode state.
in	*cipher*	A block cipher primitive.
in	*cipher_state*	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*)

Performs a deep copy of one state into another.

Parameters

in	*mode*	A block mode primitive.
in	*cipher*	A block cipher primitive.
out	*dst*	The destination state.
in	*src*	The source state.

Remarks

The destination state must have been allocated, by using the same primitive(s) as the source state, and mustn't be initialized.
The source state must be initialized.

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	*mode*	A block mode primitive.
in	*cipher*	A block cipher primitive.
in	*query*	A query code.
in	*value*	A suggested value.

Returns

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

See Also

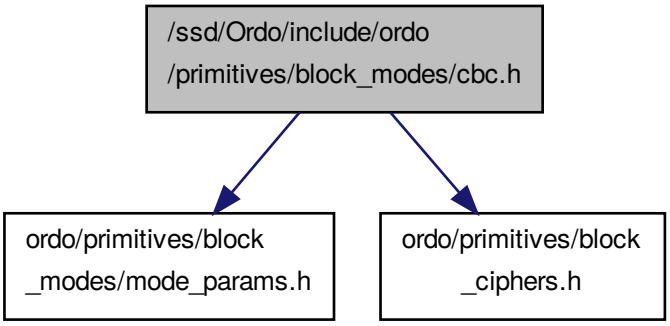
query.h

6.23 /ssd/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

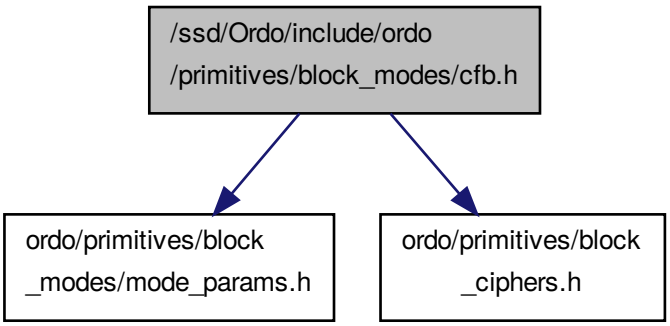
block_mode_query()

6.24 /ssd/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

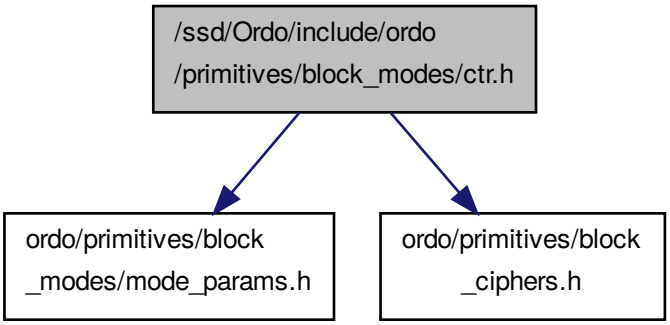
block_mode_query()

6.25 /ssd/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

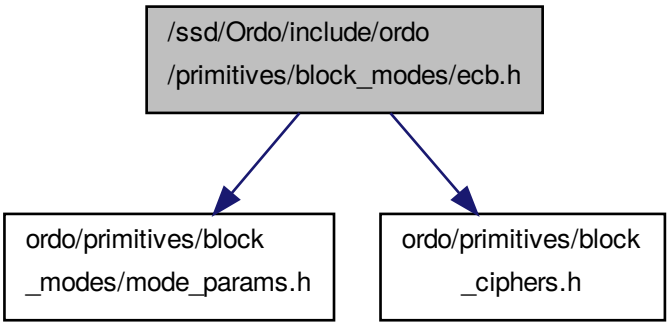
block_mode_query()

6.26 /ssd/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*)

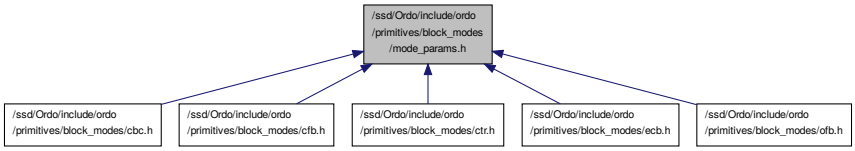
See Also

block_mode_query()

6.27 /ssd/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

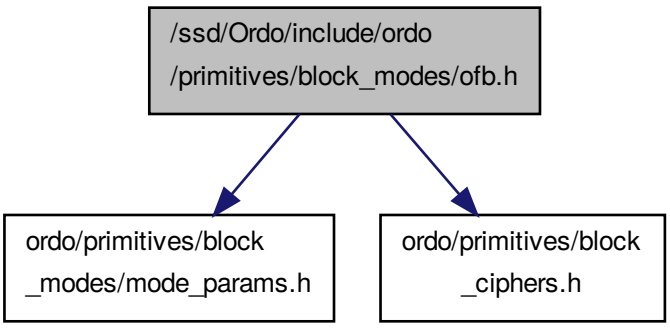
Primitive Parameters. This header contains parameter structures for all block modes.

6.28 /ssd/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*)

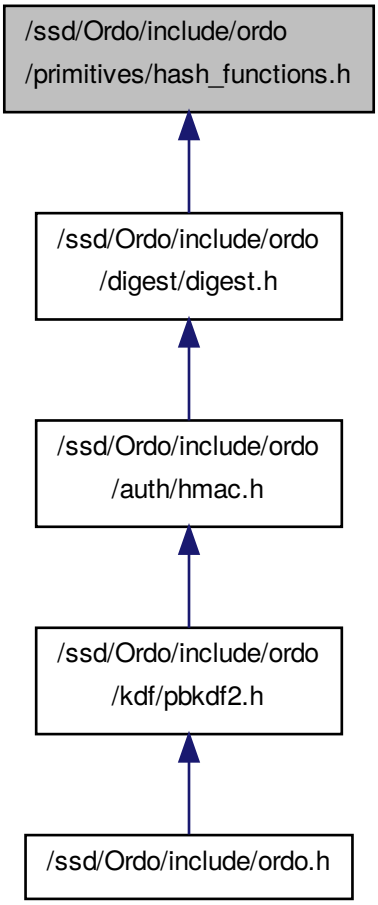
See Also

block_mode_query()

6.29 /ssd/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 * ordo_sha256 (void)

  The SHA-256 hash function.
- ORDO_PUBLIC const struct HASH_FUNCTION * ordo_md5 (void)

  The MD5 hash function.
- ORDO_PUBLIC const struct HASH_FUNCTION * ordo_skein256 (void)

  The Skein-256 hash function.
- 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 size_t hash_function_count (void)
- 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`	*primitive*	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 const struct HASH_FUNCTION* hash_function_by_name (const char * *name*)

Returns a hash function primitive from a name.

Parameters

name	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.

6.29.2.3 ORDO_PUBLIC const struct HASH_FUNCTION* hash_function_by_index (size_t *index*)

Returns a hash function primitive from an index.

Parameters

in	*index*	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).

6.29.2.4 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.

6.29.2.5 ORDO_PUBLIC void* hash_function_alloc (const struct HASH_FUNCTION * *primitive*)

Allocates a hash function state.

Parameters

in	*primitive*	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	*primitive*	A hash function primitive.
in,out	*state*	An allocated hash function state.
in	*params*	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	*primitive*	A hash function primitive.
in,out	*state*	A hash function state.
in	*buffer*	A buffer to append to the message.
in	*len*	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	*primitive*	A hash function primitive.
in,out	*state*	A hash function state.
out	*digest*	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	*primitive*	A hash function primitive.
in,out	*state*	A hash function state.

6.29.2.10 ORDO_PUBLIC void hash_function_copy (const struct HASH_FUNCTION * *primitive,* void * *dst,* const void * *src*)

Performs a deep copy of one state into another.

Parameters

in	*primitive*	A hash function primitive.
out	*dst*	The destination state.
in	*src*	The source state.

Remarks

The destination state must have been allocated, by using the same primitive(s) as the source state, and mustn't be initialized.
The source state must be initialized.

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`	*primitive*	A hash function primitive.
`in`	*query*	A query code.
`in`	*value*	A suggested value.

Returns

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

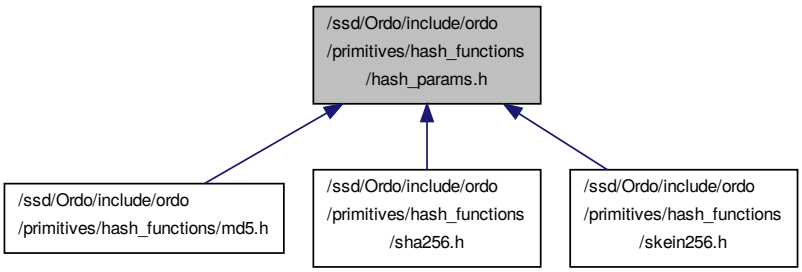
See Also

query.h

6.30 /ssd/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

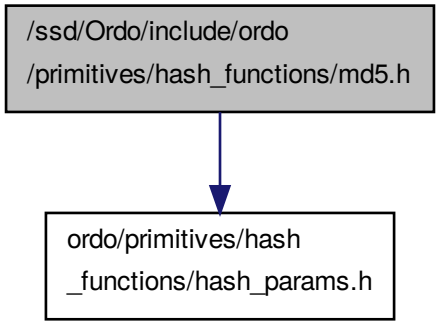
Primitive Parameters. This header contains parameter structures for all hash functions.

6.31 /ssd/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

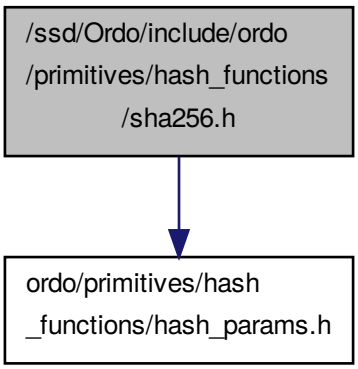
`hash_function_query()`

6.32 /ssd/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*)

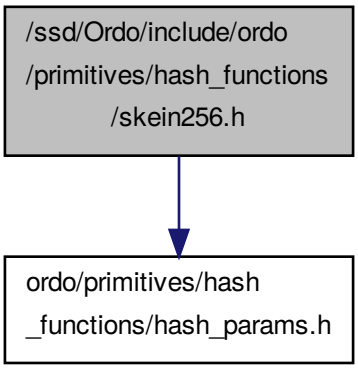
See Also

`hash_function_query()`

6.33 /ssd/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

ORDO_ARG	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*)

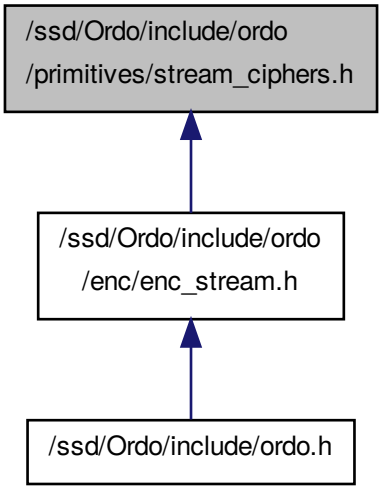
See Also

`hash_function_query()`

6.34 /ssd/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 * ordo_rc4 (void)

  The RC4 stream cipher.
- 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 size_t stream_cipher_count (void)
- 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

in	*primitive*	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 const struct STREAM_CIPHER* stream_cipher_by_name (const char * *name*)

Returns a stream cipher primitive from a name.

Parameters

name	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.

6.34.2.3 ORDO_PUBLIC const struct STREAM_CIPHER* stream_cipher_by_index (size_t *index*)

Returns a stream cipher primitive from an index.

Parameters

in	*index*	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).

6.34.2.4 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.

6.34.2.5 ORDO_PUBLIC void* stream_cipher_alloc (const struct STREAM_CIPHER * *primitive*)

Allocates a stream cipher state.

Parameters

in	*primitive*	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	*primitive*	A stream cipher primitive.
in,out	*state*	A stream cipher state.
in	*key*	The cryptographic key to use.
in	*key_len*	The length, in bytes, of the key.
in	*params*	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	*primitive*	A stream cipher primitive.
in,out	*state*	A stream cipher state.
in,out	*buffer*	The buffer to encrypt or decrypt.

in	*len*	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

in	*primitive*	A stream cipher primitive.
in,out	*state*	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

in	*primitive*	A stream cipher primitive.
in,out	*state*	A stream cipher state.

6.34.2.10 ORDO_PUBLIC void stream_cipher_copy (const struct STREAM_CIPHER * *primitive,* void * *dst,* const void * *src*)

Performs a deep copy of one state into another.

Parameters

in	*primitive*	A stream cipher primitive.
out	*dst*	The destination state.
in	*src*	The source state.

Remarks

The destination state must have been allocated, by using the same primitive(s) as the source state, and mustn't be initialized.
The source state must be initialized.

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

in	*primitive*	A stream cipher primitive.
in	*query*	A query code.
in	*value*	A suggested value.

Returns

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

See Also

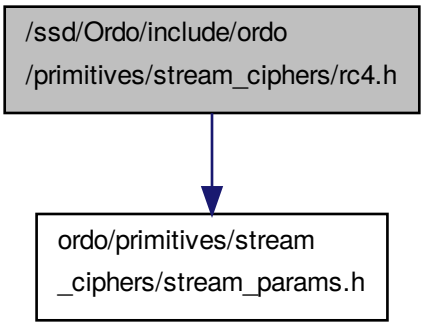
query.h

6.35 /ssd/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

ORDO_KEY_LEN	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*)

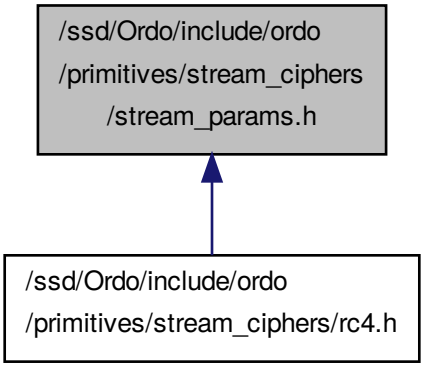
See Also

stream_cipher_query()

6.36 /ssd/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