liboqs-cpp

0.1

Generated by Doxygen 1.8.14

Contents

Chapter 1

liboqs-cpp

C++ bindings for liboqs

Build status:

liboqs-cpp offers a C++ wrapper for the Open Quantum Safe liboqs C library. The wrapper is written in standard C++11.

Contents

liboqs-cpp is a header-only wrapper. The project contains the following files and folders:

- **doc**: Doxygen-generated detailed documentation
- **examples/**kem.cpp: key encapsulation example
- **examples/**sig.cpp: signature example
- **include/oqs_cpp.h: main header file for the wrapper**
- **unit_tests**: unit tests written using Google Test (included)

Usage

To avoid namespace pollution, liboqs-cpp includes all of its code inside the namespace oqs. All of liboqs pure C API is located in the namespace oqs::C, hence to use directly a C API function the user must qualify the call with oqs::C::liboqs_C_function(...).

liboqs-cpp defines four main classes: oqs::KeyEncapsulation and oqs::Signature, providing post-quantum key encapsulation and signture mechanisms, respectively, and oqs::KEMs and oqs::Sigs, containing only static member functions that provide information related to the available key encapsulation mechanisms or signature mechanism, respectively.

oqs::KeyEncapsulation and/or oqs::Signature must be instantiated with a string identifying one of mechanisms supported by liboqs; these can be enumerated using the oqs::KEMs::get_enabled_KEM_↩ mechanisms() and oqs::Sigs::get_enabled_sig_mechanisms() member functions.

The wrapper also defines a high resolution timing class, oqs::Timer<>.

The examples in the examples folder are self-explanatory and provide more details about the wrapper's API.

liboqs installation

liboqs-cpp depends on the `liboqs` C library; liboqs must be compiled as a Linux/macOS library or as a Windows DLL, and be visible to the wrapper, e.g. installed in a system-wide folder.

Compiling on UNIX-like platforms

To use the wrapper, the user must have access to a C++11 compliant compiler, then simply `#include "oqs_cpp.h"` in her/his program. The wrapper contains a CMake build system for both examples and unit tests. To compile and run the examples, create a `build` folder inside the root folder of the project, change directory to `build`, then type

```
cmake ..; make -j 4
```

The above commands build all examples in `examples`, i.e. `examples/kem` and `examples/sig`, assuming the CMake build system is available on the user's platform. Replace the `-j 4` flag with your processor's number of cores, e.g. use `-j 8` if your system has 8 cores. To build only a specific example, e.g. `examples/kem`, specify the target as the argument of the `make` command, such as

```
make kem
```

To compile and run the unit tests, first `cd unit_tests`, then create a `build` folder inside `unit_tests`, change directory to it, and finally type

```
cmake ..; make -j 4
```

The above commands build `tests/oqs_cpp_testing` suite of unit tests.

liboqs-cpp has been extensively tested on Linux and macOS systems. Continuous integration is provided via Travis CI.

Compiling on Windows

A Visual Studio solution will be provided soon.

Limitations and security

liboqs is designed for prototyping and evaluating quantum-resistant cryptography. Security of proposed quantum-resistant algorithms may rapidly change as research advances, and may ultimately be completely insecure against either classical or quantum computers.

We believe that the NIST Post-Quantum Cryptography standardization project is currently the best avenue to identifying potentially quantum-resistant algorithms. liboqs does not intend to "pick winners", and we strongly recommend that applications and protocols rely on the outcomes of the NIST standardization project when deploying post-quantum cryptography.

We acknowledge that some parties may want to begin deploying post-quantum cryptography prior to the conclusion of the NIST standardization project. We strongly recommend that any attempts to do make use of so-called **hybrid cryptography**, in which post-quantum public-key algorithms are used alongside traditional public key algorithms (like RSA or elliptic curves) so that the solution is at least no less secure than existing traditional cryptography.

Just like liboqs, liboqs-cpp is provided "as is", without warranty of any kind. See `LICENSE` for the full disclaimer.

License

liboqs-cpp is licensed under the MIT License; see `LICENSE` for details.

Team

The Open Quantum Safe project is led by `Douglas Stebila` and `Michele Mosca` at the University of Waterloo.

liboqs-cpp was developed by `Vlad Gheorghiu` at evolutionQ and University of Waterloo.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 internal Namespace Reference

Internal implementation details.

6.1.1 Detailed Description

Internal implementation details.

6.2 oqs Namespace Reference

Main namespace for the liboqs C++ wrapper.

Namespaces

- C

  Namespace containing all of the oqs C functions, so we they do not pollute the oqs namespace.
- internal

Classes

- class KEMs

  Singleton class, contains details about supported/enabled key exchange mechanisms (KEMs)
- class KeyEncapsulation

  Key encapsulation mechanisms.
- class MechanismNotEnabledError

  Cryptographic scheme not enabled.
- class MechanismNotSupportedError

  Cryptographic scheme not supported.
- class Signature

  Signature mechanisms.
- class Sigs

  Singleton class, contains details about supported/enabled signature mechanisms.
- class Timer

  High resolution timer.

Typedefs

- using byte = std::uint8_t

  byte (unsigned)
- using bytes = std::vector< byte >

  vector of bytes (unsigned)
- using OQS_STATUS = C::OQS_STATUS

  bring OQS_STATUS into the oqs namespace

Functions

- internal::HexChop hex_chop (const oqs::bytes &v, std::size_t start=8, std::size_t end=8)

  Constructs an instance of oqs::internal::HexChop.

6.2.1 Detailed Description

Main namespace for the liboqs C++ wrapper.

6.2.2 Typedef Documentation

6.2.2.1 byte

```
using oqs::byte = typedef std::uint8_t
```

byte (unsigned)

6.2.2.2 bytes

```
using oqs::bytes = typedef std::vector<byte>
```

vector of bytes (unsigned)

6.2.2.3 OQS_STATUS

```
using oqs::OQS_STATUS = typedef C::OQS_STATUS
```

bring OQS_STATUS into the oqs namespace

6.2.3 Function Documentation

6.2.3.1 hex_chop()

```
internal::HexChop oqs::hex_chop (
             const oqs::bytes & v,
             std::size_t start = 8,
             std::size_t end = 8 )  [inline]
```

Constructs an instance of oqs::internal::HexChop.

Parameters

v	Vector of bytes
start	Number of hex characters displayed from the beginning of the vector
end	Number of hex characters displayed from the end of the vector

Returns

Instance of oqs::internal::HexChop

6.3 oqs::C Namespace Reference

Namespace containing all of the oqs C functions, so we they do not pollute the oqs namespace.

6.3.1 Detailed Description

Namespace containing all of the oqs C functions, so we they do not pollute the oqs namespace.

6.4 oqs::internal Namespace Reference

Classes

- class HexChop

  std::ostream manipulator for long vectors of oqs::byte, use it to display only a small number of elements from the beginning and end of the vector
- class Singleton

  Singleton class using CRTP pattern.

6.5 oqs_literals Namespace Reference

Functions

- oqs::bytes operator""_bytes (const char ∗c_str, std::size_t length)

  User-defined literal operator for converting C-style strings to oqs::bytes.

6.5.1 Function Documentation

6.5.1.1 operator""""_bytes()

```
oqs::bytes oqs_literals::operator""_bytes (
             const char * c_str,
             std::size_t length )  [inline]
```

User-defined literal operator for converting C-style strings to oqs::bytes.

Note

The null terminator is not included

Parameters

c_str	C-style string
length	C-style string length (deduced automatically by the compiler)

Returns

The byte representation of the input C-style string

Chapter 7

Class Documentation

7.1 oqs::KeyEncapsulation::alg_details_ Struct Reference

KEM algorithm details.

Public Attributes

- std::string name
- std::string version
- std::size_t claimed_nist_level
- bool is_ind_cca
- std::size_t length_public_key
- std::size_t length_secret_key
- std::size_t length_ciphertext
- std::size_t length_shared_secret

7.1.1 Detailed Description

KEM algorithm details.

7.1.2 Member Data Documentation

7.1.2.1 claimed_nist_level

```
std::size_t oqs::KeyEncapsulation::alg_details_::claimed_nist_level
```

7.1.2.2 is_ind_cca

```
bool oqs::KeyEncapsulation::alg_details_::is_ind_cca
```

7.1.2.3 length_ciphertext

```
std::size_t oqs::KeyEncapsulation::alg_details_::length_ciphertext
```

7.1.2.4 length_public_key

```
std::size_t oqs::KeyEncapsulation::alg_details_::length_public_key
```

7.1.2.5 length_secret_key

```
std::size_t oqs::KeyEncapsulation::alg_details_::length_secret_key
```

7.1.2.6 length_shared_secret

```
std::size_t oqs::KeyEncapsulation::alg_details_::length_shared_secret
```

7.1.2.7 name

```
std::string oqs::KeyEncapsulation::alg_details_::name
```

7.1.2.8 version

```
std::string oqs::KeyEncapsulation::alg_details_::version
```

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

- oqs_cpp.h

7.2 oqs::Signature::alg_details_ Struct Reference

Signature algorithm details.

Public Attributes

- std::string name
- std::string version
- std::size_t claimed_nist_level
- bool is_euf_cma
- std::size_t length_public_key
- std::size_t length_secret_key
- std::size_t length_signature

7.2.1 Detailed Description

Signature algorithm details.

7.2.2 Member Data Documentation

7.2.2.1 claimed_nist_level

```
std::size_t oqs::Signature::alg_details_::claimed_nist_level
```

7.2.2.2 is_euf_cma

```
bool oqs::Signature::alg_details_::is_euf_cma
```

7.2.2.3 length_public_key

```
std::size_t oqs::Signature::alg_details_::length_public_key
```

7.2.2.4 length_secret_key

```
std::size_t oqs::Signature::alg_details_::length_secret_key
```

7.2.2.5 length_signature

```
std::size_t oqs::Signature::alg_details_::length_signature
```

7.2.2.6 name

```
std::string oqs::Signature::alg_details_::name
```

7.2.2.7 version

```
std::string oqs::Signature::alg_details_::version
```

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

- oqs_cpp.h

7.3 oqs::internal::HexChop Class Reference

std::ostream manipulator for long vectors of oqs::byte, use it to display only a small number of elements from the beginning and end of the vector

```
#include <oqs_cpp.h>
```

Public Member Functions

- HexChop (const oqs::bytes &v, std::size_t start, std::size_t end)

  Constructs an instance of oqs::internal::HexChop.

Private Member Functions

- void manipulate_ostream_ (std::ostream &os, std::size_t start, std::size_t end, bool is_short) const

  std::ostream manipulator

Private Attributes

- bytes v_

  vector of byes
- std::size_t start_
- std::size_t end_

  number of hex bytes taken from the start and from the end

Friends

- std::ostream & operator<< (std::ostream &os, const HexChop &rhs)

  std::ostream extraction operator for oqs::internal::HexChop

7.3.1 Detailed Description

std::ostream manipulator for long vectors of oqs::byte, use it to display only a small number of elements from the beginning and end of the vector

7.3.2 Constructor & Destructor Documentation

7.3.2.1 HexChop()

```
oqs::internal::HexChop::HexChop (
            const oqs::bytes & v,
            std::size_t start,
            std::size_t end )  [inline], [explicit]
```

Constructs an instance of oqs::internal::HexChop.

Parameters

v	Vector of bytes
start	Number of hex characters displayed from the beginning of the vector
end	Number of hex characters displayed from the end of the vector

7.3.3 Member Function Documentation

7.3.3.1 manipulate_ostream_()

```
void oqs::internal::HexChop::manipulate_ostream_ (
            std::ostream & os,
            std::size_t start,
            std::size_t end,
            bool is_short ) const  [inline], [private]
```

std::ostream manipulator

Parameters

os	Output stream
start	Number of hex characters displayed from the beginning of the vector
end	Number of hex characters displayed from the end of the vector
is_short	Vector is too short, display all hex characters

7.3.4 Friends And Related Function Documentation

7.3.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const HexChop & rhs )  [friend]
```

std::ostream extraction operator for oqs::internal::HexChop

Parameters

os	Output stream
rhs	oqs::internal::HexChop instance

Returns

Reference to the output stream

7.3.5 Member Data Documentation

7.3.5.1 end_

```
std::size_t oqs::internal::HexChop::end_  [private]
```

number of hex bytes taken from the start and from the end

7.3.5.2 start_

```
std::size_t oqs::internal::HexChop::start_  [private]
```

7.3.5.3 v_

```
bytes oqs::internal::HexChop::v_  [private]
```

vector of byes

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

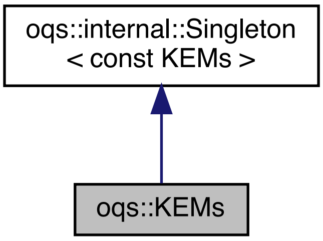
- oqs_cpp.h

7.4 oqs::KEMs Class Reference

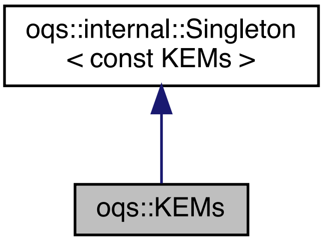
Singleton class, contains details about supported/enabled key exchange mechanisms (KEMs)

```
#include <oqs_cpp.h>
```

Inheritance diagram for oqs::KEMs:



Collaboration diagram for oqs::KEMs:



Static Public Member Functions

- static std::size_t max_number_KEMs ()

  Maximum number of supported KEMs.
- static bool is_KEM_supported (const std::string &alg_name)

  Checks whether the KEM algorithm alg_name is supported.
- static bool is_KEM_enabled (const std::string &alg_name)

  Checks whether the KEM algorithm alg_name is enabled.
- static std::string get_KEM_name (std::size_t alg_id)

  KEM algorithm name.
- static const std::vector< std::string > & get_supported_KEMs ()

  Vector of supported KEM algorithms.
- static const std::vector< std::string > & get_enabled_KEMs ()

  Vector of enabled KEM algorithms.

Private Member Functions

- KEMs ()=default

  Private default constructor.

Friends

- class internal::Singleton< const KEMs >

Additional Inherited Members

7.4.1 Detailed Description

Singleton class, contains details about supported/enabled key exchange mechanisms (KEMs)

7.4.2 Constructor & Destructor Documentation

7.4.2.1 KEMs()

```
oqs::KEMs::KEMs ( )  [private], [default]
```

Private default constructor.

Note

Use oqs::KEMs::get_instance() to create an instance

7.4.3 Member Function Documentation

7.4.3.1 get_enabled_KEMs()

```
static const std::vector<std::string>& oqs::KEMs::get_enabled_KEMs ( )  [inline], [static]
```

Vector of enabled KEM algorithms.

Returns

Vector of enabled KEM algorithms

7.4.3.2 get_KEM_name()

```
static std::string oqs::KEMs::get_KEM_name (
             std::size_t alg_id )  [inline], [static]
```

KEM algorithm name.

Parameters

alg← _id	Cryptographic algorithm numerical id

Returns

KEM algorithm name

7.4.3.3 get_supported_KEMs()

```
static const std::vector<std::string>& oqs::KEMs::get_supported_KEMs ( )  [inline], [static]
```

Vector of supported KEM algorithms.

Returns

Vector of supported KEM algorithms

7.4.3.4 is_KEM_enabled()

```
static bool oqs::KEMs::is_KEM_enabled (
             const std::string & alg_name )  [inline], [static]
```

Checks whether the KEM algorithm *alg_name* is enabled.

Parameters

alg_name	Cryptographic algorithm name

Returns

True if the KEM algorithm is enabled, false otherwise

7.4.3.5 is_KEM_supported()

```
static bool oqs::KEMs::is_KEM_supported (
             const std::string & alg_name )  [inline], [static]
```

Checks whether the KEM algorithm *alg_name* is supported.

Parameters

alg_name	Cryptographic algorithm name

Returns

True if the KEM algorithm is supported, false otherwise

7.4.3.6 max_number_KEMs()

```
static std::size_t oqs::KEMs::max_number_KEMs ( )  [inline], [static]
```

Maximum number of supported KEMs.

Returns

Maximum number of supported KEMs

7.4.4 Friends And Related Function Documentation

7.4.4.1 internal::Singleton< const KEMs >

```
friend class internal::Singleton< const KEMs >  [friend]
```

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

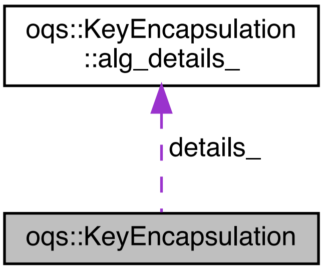
- oqs_cpp.h

7.5 oqs::KeyEncapsulation Class Reference

Key encapsulation mechanisms.

```
#include <oqs_cpp.h>
```

Collaboration diagram for oqs::KeyEncapsulation:



Classes

- struct alg_details_

    KEM algorithm details.

Public Member Functions

- KeyEncapsulation (const std::string &alg_name, const bytes &secret_key={})

    Constructs an instance of oqs::KeyEncapsulation.
- virtual ∼KeyEncapsulation ()

    Virtual default destructor.
- const alg_details_ & get_details () const

    KEM algorithm details.
- bytes generate_keypair ()

    Generate public key/secret key pair.
- bytes export_secret_key () const

    Export secret key.
- std::pair< bytes, bytes > encap_secret (const bytes &public_key) const

    Encapsulate secret.
- bytes decap_secret (const bytes &ciphertext) const

    Decapsulate secret.

Private Attributes

- const std::string alg_name_

    cryptographic algorithm name
- std::shared_ptr< C::OQS_KEM > kem_

    liboqs smart pointer to C::OQS_KEM
- bytes secret_key_ {}

    secret key
- struct oqs::KeyEncapsulation::alg_details_ details_

Friends

- std::ostream & operator<< (std::ostream &os, const alg_details_ &rhs)

    std::ostream extraction operator for the KEM algorithm details
- std::ostream & operator<< (std::ostream &os, const KeyEncapsulation &rhs)

    std::ostream extraction operator for oqs::KeyEncapsulation

7.5.1 Detailed Description

Key encapsulation mechanisms.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 KeyEncapsulation()

```
oqs::KeyEncapsulation::KeyEncapsulation (
            const std::string & alg_name,
            const bytes & secret_key = {} )  [inline], [explicit]
```

Constructs an instance of oqs::KeyEncapsulation.

Parameters

alg_name	Cryptographic algorithm name
secret_key	Secret key (optional)

7.5.2.2 ∼KeyEncapsulation()

```
virtual oqs::KeyEncapsulation::∼KeyEncapsulation ( )  [inline], [virtual]
```

Virtual default destructor.

7.5.3 Member Function Documentation

7.5.3.1 decap_secret()

```
bytes oqs::KeyEncapsulation::decap_secret (
             const bytes & ciphertext ) const  [inline]
```

Decapsulate secret.

Parameters

ciphertext	Ciphertext

Returns

Shared secret

7.5.3.2 encap_secret()

```
std::pair<bytes, bytes> oqs::KeyEncapsulation::encap_secret (
             const bytes & public_key ) const  [inline]
```

Encapsulate secret.

Parameters

public_key	Public key

Returns

Pair consisting of 1) ciphertext, and 2) shared secret

7.5.3.3 export_secret_key()

```
bytes oqs::KeyEncapsulation::export_secret_key ( ) const  [inline]
```

Export secret key.

Returns

Secret key

7.5.3.4 generate_keypair()

```
bytes oqs::KeyEncapsulation::generate_keypair ( )  [inline]
```

Generate public key/secret key pair.

Returns

Public key

7.5.3.5 get_details()

```
const alg_details_& oqs::KeyEncapsulation::get_details ( ) const  [inline]
```

KEM algorithm details.

Returns

KEM algorithm details

7.5.4 Friends And Related Function Documentation

7.5.4.1 operator<< [1/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const alg_details_ & rhs )  [friend]
```

std::ostream extraction operator for the KEM algorithm details

Parameters

os	Output stream
rhs	Algorithm details instance

Returns

Reference to the output stream

7.5.4.2 operator<< [2/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const KeyEncapsulation & rhs )  [friend]
```

std::ostream extraction operator for oqs::KeyEncapsulation

Parameters

os	Output stream
rhs	Key encapsulation instance

Returns

Reference to the output stream

7.5.5 Member Data Documentation

7.5.5.1 alg_name_

```
const std::string oqs::KeyEncapsulation::alg_name_  [private]
```

cryptographic algorithm name

7.5.5.2 details_

```
struct oqs::KeyEncapsulation::alg_details_ oqs::KeyEncapsulation::details_  [private]
```

7.5.5.3 kem_

```
std::shared_ptr<C::OQS_KEM> oqs::KeyEncapsulation::kem_ [private]
```

Initial value:

```
{nullptr, [](C::OQS_KEM* p) {
                                       C::OQS_KEM_free(p);
                                   }}
```

liboqs smart pointer to C::OQS_KEM

7.5.5.4 secret_key_

```
bytes oqs::KeyEncapsulation::secret_key_ {} [private]
```

secret key

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

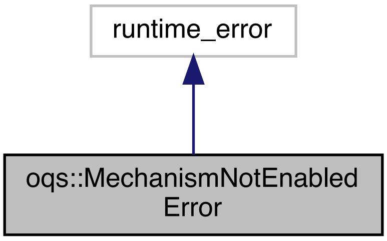
- oqs_cpp.h

7.6 oqs::MechanismNotEnabledError Class Reference

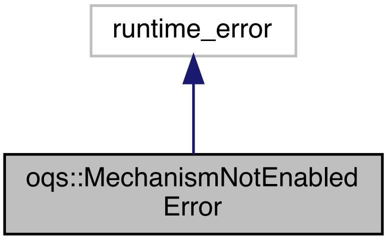
Cryptographic scheme not enabled.

```
#include <oqs_cpp.h>
```

Inheritance diagram for oqs::MechanismNotEnabledError:



Collaboration diagram for oqs::MechanismNotEnabledError:



Public Member Functions

- MechanismNotEnabledError (const std::string &alg_name)

  Constructor.

7.6.1 Detailed Description

Cryptographic scheme not enabled.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 MechanismNotEnabledError()

```
oqs::MechanismNotEnabledError::MechanismNotEnabledError (
            const std::string & alg_name )  [inline], [explicit]
```

Constructor.

Parameters

alg_name	Cryptographic algorithm name

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

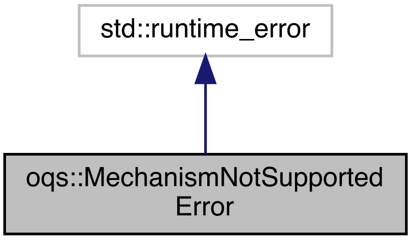
- oqs_cpp.h

7.7 oqs::MechanismNotSupportedError Class Reference

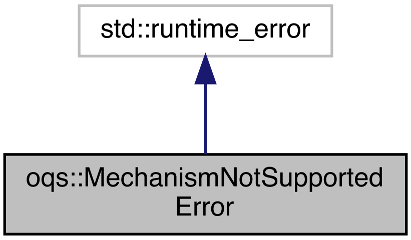
Cryptographic scheme not supported.

```
#include <oqs_cpp.h>
```

Inheritance diagram for oqs::MechanismNotSupportedError:



Collaboration diagram for oqs::MechanismNotSupportedError:



Public Member Functions

- MechanismNotSupportedError (const std::string &alg_name)

  Constructor.

7.7.1 Detailed Description

Cryptographic scheme not supported.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 MechanismNotSupportedError()

```
oqs::MechanismNotSupportedError::MechanismNotSupportedError (
            const std::string & alg_name )  [inline], [explicit]
```

Constructor.

Parameters

alg_name	Cryptographic algorithm name

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

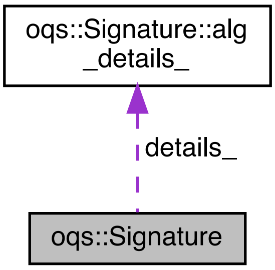
- oqs_cpp.h

7.8 oqs::Signature Class Reference

Signature mechanisms.

```
#include <oqs_cpp.h>
```

Collaboration diagram for oqs::Signature:



Classes

- struct alg_details_

  Signature algorithm details.

Public Member Functions

- Signature (const std::string &alg_name, const bytes &secret_key={})

  Constructs an instance of oqs::Signature.
- virtual ∼Signature ()

  Virtual default destructor.
- const alg_details_ & get_details () const

  Signature algorithm details.
- bytes generate_keypair ()

  Generate public key/secret key pair.
- bytes export_secret_key () const

  Export secret key.
- bytes sign (const bytes &message) const

  Sign message.
- bool verify (const bytes &message, const bytes &signature, const bytes &public_key) const

  Verify signature.

Private Attributes

- const std::string alg_name_

  cryptographic algorithm name
- std::shared_ptr< C::OQS_SIG > sig_

  liboqs smart pointer to C::OQS_SIG
- bytes secret_key_ {}

  secret key
- struct oqs::Signature::alg_details_ details_

Friends

- std::ostream & operator<< (std::ostream &os, const alg_details_ &rhs)

  std::ostream extraction operator for the signature algorithm details
- std::ostream & operator<< (std::ostream &os, const Signature &rhs)

  std::ostream extraction operator for oqs::Signature

7.8.1 Detailed Description

Signature mechanisms.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 Signature()

```
oqs::Signature::Signature (
            const std::string & alg_name,
            const bytes & secret_key = {} )  [inline], [explicit]
```

Constructs an instance of oqs::Signature.

Parameters

alg_name	Cryptographic algorithm name
secret_key	Secret key (optional)

7.8.2.2 ∼Signature()

```
virtual oqs::Signature::∼Signature ( )  [inline], [virtual]
```

Virtual default destructor.

7.8.3 Member Function Documentation

7.8.3.1 export_secret_key()

```
bytes oqs::Signature::export_secret_key ( ) const  [inline]
```

Export secret key.

Returns

Secret key

7.8.3.2 generate_keypair()

```
bytes oqs::Signature::generate_keypair ( )  [inline]
```

Generate public key/secret key pair.

Returns

Public key

7.8.3.3 get_details()

```
const alg_details_& oqs::Signature::get_details ( ) const  [inline]
```

Signature algorithm details.

Returns

Signature algorithm details

7.8.3.4 sign()

```
bytes oqs::Signature::sign (
             const bytes & message ) const  [inline]
```

Sign message.

Parameters

message	Message

Returns

Message signature

7.8.3.5 verify()

```
bool oqs::Signature::verify (
             const bytes & message,
             const bytes & signature,
             const bytes & public_key ) const  [inline]
```

Verify signature.

Parameters

message	Message
signature	Signature
public_key	Public key

Returns

True if the signature is valid, false otherwise

7.8.4 Friends And Related Function Documentation

7.8.4.1 operator<< [1/2]

```
std::ostream& operator<< (
             std::ostream & os,
             const alg_details_ & rhs )  [friend]
```

std::ostream extraction operator for the signature algorithm details

Parameters

os	Output stream
rhs	Algorithm details

Returns

Reference to the output stream

7.8.4.2 operator<< [2/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const Signature & rhs )  [friend]
```

std::ostream extraction operator for oqs::Signature

Parameters

os	Output stream
rhs	Signature instance

Returns

Reference to the output stream

7.8.5 Member Data Documentation

7.8.5.1 alg_name_

```
const std::string oqs::Signature::alg_name_  [private]
```

cryptographic algorithm name

7.8.5.2 details_

```
struct oqs::Signature::alg_details_ oqs::Signature::details_  [private]
```

7.8.5.3 secret_key_

```
bytes oqs::Signature::secret_key_ {}  [private]
```

secret key

7.8.5.4 sig_

```
std::shared_ptr<C::OQS_SIG> oqs::Signature::sig_  [private]
```

Initial value:

```
{nullptr, [](C::OQS_SIG* p) {
                                        C::OQS_SIG_free(p);
                                    }}
```

liboqs smart pointer to C::OQS_SIG

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

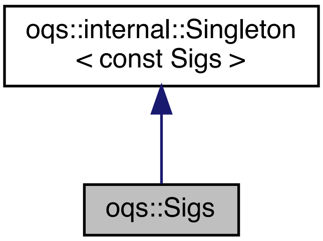
- oqs_cpp.h

7.9 oqs::Sigs Class Reference

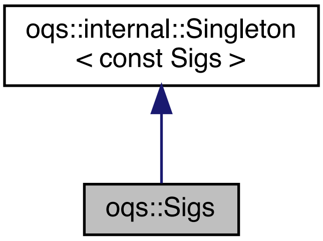
Singleton class, contains details about supported/enabled signature mechanisms.

```
#include <oqs_cpp.h>
```

Inheritance diagram for oqs::Sigs:



Collaboration diagram for oqs::Sigs:



Static Public Member Functions

- static std::size_t max_number_sigs ()

  Maximum number of supported signatures.
- static bool is_sig_supported (const std::string &alg_name)

  Checks whether the signature algorithm alg_name is supported.
- static bool is_sig_enabled (const std::string &alg_name)

  Checks whether the signature algorithm alg_name is enabled.
- static std::string get_sig_name (std::size_t alg_id)

  Signature algorithm name.
- static const std::vector< std::string > & get_supported_sigs ()

  Vector of supported signature algorithms.
- static const std::vector< std::string > & get_enabled_sigs ()

  Vector of enabled signature algorithms.

Private Member Functions

- Sigs ()=default

  Private default constructor.

Friends

- class internal::Singleton< const Sigs >

Additional Inherited Members

7.9.1 Detailed Description

Singleton class, contains details about supported/enabled signature mechanisms.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Sigs()

```
oqs::Sigs::Sigs ( )  [private], [default]
```

Private default constructor.

Note

> Use oqs::Sigs::get_instance() to create an instance

7.9.3 Member Function Documentation

7.9.3.1 get_enabled_sigs()

```
static const std::vector<std::string>& oqs::Sigs::get_enabled_sigs ( )  [inline], [static]
```

Vector of enabled signature algorithms.

Returns

Vector of enabled signature algorithms

7.9.3.2 get_sig_name()

```
static std::string oqs::Sigs::get_sig_name (
             std::size_t alg_id )  [inline], [static]
```

Signature algorithm name.

Parameters

alg↩ _id	Cryptographic algorithm numerical id

Returns

Signature algorithm name

7.9.3.3 get_supported_sigs()

```
static const std::vector<std::string>& oqs::Sigs::get_supported_sigs ( )  [inline], [static]
```

Vector of supported signature algorithms.

Returns

Vector of supported signature algorithms

7.9.3.4 is_sig_enabled()

```
static bool oqs::Sigs::is_sig_enabled (
             const std::string & alg_name )  [inline], [static]
```

Checks whether the signature algorithm *alg_name* is enabled.

Parameters

alg_name	Cryptographic algorithm name

Returns

True if the signature algorithm is enabled, false otherwise

7.9.3.5 is_sig_supported()

```
static bool oqs::Sigs::is_sig_supported (
            const std::string & alg_name )  [inline], [static]
```

Checks whether the signature algorithm *alg_name* is supported.

Parameters

alg_name	Cryptographic algorithm name

Returns

True if the signature algorithm is supported, false otherwise

7.9.3.6 max_number_sigs()

```
static std::size_t oqs::Sigs::max_number_sigs ( )  [inline], [static]
```

Maximum number of supported signatures.

Returns

Maximum number of supported signatures

7.9.4 Friends And Related Function Documentation

7.9.4.1 internal::Singleton< const Sigs >

```
friend class internal::Singleton< const Sigs >  [friend]
```

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

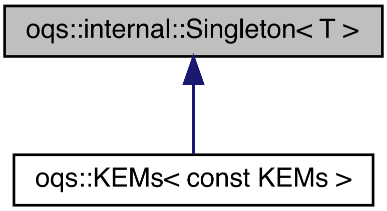
- oqs_cpp.h

7.10 oqs::internal::Singleton< T > Class Template Reference

Singleton class using CRTP pattern.

```
#include <oqs_cpp.h>
```

Inheritance diagram for oqs::internal::Singleton< T >:



Static Public Member Functions

- static T & get_instance () noexcept(std::is_nothrow_constructible< T >::value)

  Singleton instance (thread-safe) via CRTP pattern.

Protected Member Functions

- Singleton () noexcept=default
- Singleton (const Singleton &)=delete
- Singleton & operator= (const Singleton &)=delete
- virtual ∼Singleton ()=default

7.10.1 Detailed Description

template<typename T>
class oqs::internal::Singleton< T >

Singleton class using CRTP pattern.

Note

Code from `https://github.com/vsoftco/qpp/blob/master/include/internal/classes/singleton h`

Template Parameters

T	Class type of which instance will become a Singleton

7.10.2 Constructor & Destructor Documentation

7.10.2.1 Singleton() [1/2]

```
template<typename T>
oqs::internal::Singleton< T >::Singleton ( )  [protected], [default], [noexcept]
```

7.10.2.2 Singleton() [2/2]

```
template<typename T>
oqs::internal::Singleton< T >::Singleton (
             const Singleton< T > &  )  [protected], [delete]
```

7.10.2.3 ∼Singleton()

```
template<typename T>
virtual oqs::internal::Singleton< T >::∼Singleton ( )  [protected], [virtual], [default]
```

7.10.3 Member Function Documentation

7.10.3.1 get_instance()

```
template<typename T>
static T& oqs::internal::Singleton< T >::get_instance ( )  [inline], [static], [noexcept]
```

Singleton instance (thread-safe) via CRTP pattern.

Returns

Singleton instance

7.10.3.2 operator=()

```
template<typename T>
Singleton& oqs::internal::Singleton< T >::operator= (
             const Singleton< T > &  )  [protected], [delete]
```

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

- oqs_cpp.h

7.11 oqs::Timer< T, CLOCK_T > Class Template Reference

High resolution timer.

```
#include <oqs_cpp.h>
```

Public Member Functions

- Timer () noexcept

  Constructs an instance with the current time as the start point.
- void tic () noexcept

  Resets the chronometer.
- const Timer & toc () &noexcept

  Stops the chronometer.
- double tics () const noexcept

  Time passed in the duration specified by T.
- template<typename U = T>
  U get_duration () const noexcept

  Duration specified by U.
- virtual ∼Timer ()=default

  Default virtual destructor.

Protected Attributes

- CLOCK_T::time_point start_
- CLOCK_T::time_point end_

Friends

- std::ostream & operator<< (std::ostream &os, const Timer &rhs)

7.11.1 Detailed Description

template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady_clock>
class oqs::Timer< T, CLOCK_T >

High resolution timer.

Note

Code from https://github.com/vsoftco/qpp/blob/master/include/classes/timer.↩
h

Template Parameters

T	Tics duration, default is std::chrono::duration<double>, i.e. seconds in double precision
CLOCK↩ _T	Clock's type, default is std::chrono::steady_clock, not affected by wall clock changes during runtime

7.11.2 Constructor & Destructor Documentation

7.11.2.1 Timer()

```
template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady↩
_clock>
oqs::Timer< T, CLOCK_T >::Timer ( )  [inline], [noexcept]
```

Constructs an instance with the current time as the start point.

7.11.2.2 ∼Timer()

```
template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady↩
_clock>
virtual oqs::Timer< T, CLOCK_T >::∼Timer ( )  [virtual], [default]
```

Default virtual destructor.

7.11.3 Member Function Documentation

7.11.3.1 get_duration()

```
template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady↩
_clock>
template<typename U = T>
U oqs::Timer< T, CLOCK_T >::get_duration ( ) const  [inline], [noexcept]
```

Duration specified by U.

Template Parameters

U	Duration, default is T, which defaults to std::chrono::duration<double>, i.e. seconds in double precision

Returns

Duration that passed between the instantiation/reset and invocation of oqs::Timer::toc()

7.11.3.2 tic()

```
template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady↩
_clock>
void oqs::Timer< T, CLOCK_T >::tic ( )  [inline], [noexcept]
```

Resets the chronometer.

Resets the start/end point to the current time

7.11.3.3 tics()

```
template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady↩
_clock>
double oqs::Timer< T, CLOCK_T >::tics ( ) const  [inline], [noexcept]
```

Time passed in the duration specified by T.

Returns

Number of tics (specified by T) that passed between the instantiation/reset and invocation of oqs::Timer::toc()

7.11.3.4 toc()

```
template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady↩
_clock>
const Timer& oqs::Timer< T, CLOCK_T >::toc ( ) &  [inline], [noexcept]
```

Stops the chronometer.

Set the current time as the end point

Returns

Reference to the current instance

7.11.4 Friends And Related Function Documentation

7.11.4.1 operator<<

```
template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady↩
_clock>
std::ostream& operator<< (
             std::ostream & os,
             const Timer< T, CLOCK_T > & rhs )  [friend]
```

7.11.5 Member Data Documentation

7.11.5.1 end_

```
template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady↩
_clock>
CLOCK_T::time_point oqs::Timer< T, CLOCK_T >::end_ [protected]
```

7.11.5.2 start_

```
template<typename T = std::chrono::duration<double>, typename CLOCK_T = std::chrono::steady↩
_clock>
CLOCK_T::time_point oqs::Timer< T, CLOCK_T >::start_ [protected]
```

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

- oqs_cpp.h

Chapter 8

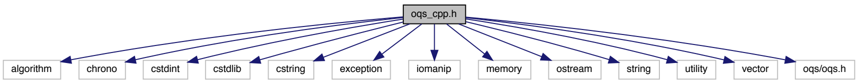
File Documentation

8.1 oqs_cpp.h File Reference

Main header file for the liboqs C++ wrapper.

```
#include <algorithm>
#include <chrono>
#include <cstdint>
#include <cstdlib>
#include <cstring>
#include <exception>
#include <iomanip>
#include <memory>
#include <ostream>
#include <string>
#include <utility>
#include <vector>
#include <oqs/oqs.h>
```

Include dependency graph for oqs_cpp.h:



Classes

- class oqs::internal::Singleton< T >

  Singleton class using CRTP pattern.
- class oqs::internal::HexChop

  std::ostream manipulator for long vectors of oqs::byte, use it to display only a small number of elements from the beginning and end of the vector
- class oqs::Timer< T, CLOCK_T >

  High resolution timer.
- class oqs::MechanismNotSupportedError

  Cryptographic scheme not supported.

- class oqs::MechanismNotEnabledError

  Cryptographic scheme not enabled.
- class oqs::KEMs

  Singleton class, contains details about supported/enabled key exchange mechanisms (KEMs)
- class oqs::KeyEncapsulation

  Key encapsulation mechanisms.
- struct oqs::KeyEncapsulation::alg_details_

  KEM algorithm details.
- class oqs::Sigs

  Singleton class, contains details about supported/enabled signature mechanisms.
- class oqs::Signature

  Signature mechanisms.
- struct oqs::Signature::alg_details_

  Signature algorithm details.

Namespaces

- oqs

  Main namespace for the liboqs C++ wrapper.
- oqs::C

  Namespace containing all of the oqs C functions, so we they do not pollute the oqs namespace.
- internal

  Internal implementation details.
- oqs::internal
- oqs_literals

Typedefs

- using oqs::byte = std::uint8_t

  byte (unsigned)
- using oqs::bytes = std::vector< byte >

  vector of bytes (unsigned)
- using oqs::OQS_STATUS = C::OQS_STATUS

  bring OQS_STATUS into the oqs namespace

Functions

- internal::HexChop oqs::hex_chop (const oqs::bytes &v, std::size_t start=8, std::size_t end=8)

  Constructs an instance of oqs::internal::HexChop.
- std::ostream & operator<< (std::ostream &os, const oqs::bytes &rhs)

  std::ostream extraction operator for oqs::bytes
- std::ostream & operator<< (std::ostream &os, const std::vector< std::string > &rhs)

  std::ostream extraction operator for vectors of strings
- oqs::bytes oqs_literals::operator""_bytes (const char ∗c_str, std::size_t length)

  User-defined literal operator for converting C-style strings to oqs::bytes.

8.1.1 Detailed Description

Main header file for the liboqs C++ wrapper.

8.1.2 Function Documentation

8.1.2.1 operator<<() [1/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const oqs::bytes & rhs )  [inline]
```

std::ostream extraction operator for oqs::bytes

Parameters

os	Output stream
rhs	Vector of oqs::byte

Returns

Reference to the output stream

8.1.2.2 operator<<() [2/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const std::vector< std::string > & rhs )  [inline]
```

std::ostream extraction operator for vectors of strings

Parameters

os	Output stream
rhs	Vector of std::string

Returns

Reference to the output stream

Index